

For Reference

NOT TO BE TAKEN FROM THIS ROOM

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex LIBRIS
UNIVERSITATIS
ALBERTAENSIS





Digitized by the Internet Archive
in 2019 with funding from
University of Alberta Libraries

<https://archive.org/details/OSBishop1969>

THE UNIVERSITY OF ALBERTA

A QUEUEING ANALYSIS OF THE CP/67 TIME-SHARING SYSTEM

by



OTHNIEL STEWART BISHOP

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1969

Shaw
1929/5
23

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and
recommend to the Faculty of Graduate Studies for acceptance,
a thesis entitled A QUEUEING ANALYSIS OF THE CP/67 TIME-SHARING
SYSTEM submitted by OTHNIEL STEWART BISHOP in partial fulfilment
of the requirements for the degree of Master of Science.

ABSTRACT

This thesis presents a study of the CP/67 Time-Sharing System using the techniques of Queueing Theory. Brief reviews are given of time-sharing systems and their properties in Chapter II, and of Queueing Theory and some of its applications to these systems in Chapter III. A description of CP/67 is provided in Chapter IV. Finally a model of its dispatcher, based on preemptive queues, is developed and analysed.

ACKNOWLEDGEMENTS

I should like to express my deepest gratitude to Professor U.M. von Maydell whose suggestions and advice were of invaluable assistance at all stages of this study. I wish to thank Professor D.B. Scott, Head of the Department of Computing Science, for providing computing facilities, and Mr. D. Webster for his enlightening talks on the CP/CMS system. In addition, I wish to thank the Canadian International Development Agency for financial assistance in carrying out this research.

TABLE OF CONTENTS

	<u>Page</u>
CHAPTER I: INTRODUCTION	
1.1 Studies and Analyses of Time-Sharing Systems	1
1.2 Type of Analysis Used in This Thesis ...	3
CHAPTER II: TIME-SHARING COMPUTER SYSTEMS	
2.1 Evolution of Time-Sharing and Some Definitions	5
2.2 Virtual Machines	7
2.3 The Operating System	9
2.4 Memory	10
2.4.1 Paging	12
2.4.2 Swapping Algorithm	13
2.5 The Supervisor	16
2.5.1 The Interrupt Processor	17
2.5.2 The I/O Processor	18
2.5.3 Timer Administrator	19
2.5.4 Program Storage Administrator ...	20
2.5.5 Facilities Administrator	20
2.5.6 Program Administrator	21
2.5.7 The Scheduler/Dispatcher	23
CHAPTER III: A SURVEY OF QUEUEING THEORY APPLICATIONS IN TIME-SHARING COMPUTER SYSTEMS	
3.1 Introduction to Queueing Theory in Time-Sharing Systems	26
3.1.1 Aspects of Priority Queues	28
3.2 Survey of Some Analytic Studies of Computer Systems	29
3.2.1 Queueing Analysis of Disk Storage Facility	30
3.2.2 Congestion Analysis of a Computer Core Storage System	33
3.2.3 Shemer's Mathematical Consideration of Time-Sharing Scheduling Algorithms	34
3.2.4 The SRPT and SPT Disciplines by Schrage	37
3.2.5 Two Algorithms by Coffman for Limited Swapping	39

CHAPTER IV:	CP/67 TIME-SHARING SYSTEM	
4.1	The CP/67 Operating System	44
4.1.1	Design Goals	44
4.1.2	Components of the System	44
4.1.3	Hardware Environment	45
4.1.4	Present Implementation at University of Alberta	46
4.2	The CP/67 Supervisor	46
4.2.1	Virtual Machines in CP/67	46
4.2.2	Description of Some Mapping Tables	48
4.2.3	Virtual Addressing	50
4.2.4	Paging in CP/67	52
4.2.5	Swapping Algorithm	53
4.3	The Dispatcher in CP/67	54
4.3.1	Users' States	55
4.3.2	Characteristics of Q_1 and Q_2 Users	57
CHAPTER V:	A QUEUEING MODEL OF THE CP/67 DISPATCHER	
5.1	Formulation of the Model	61
5.2	Analysis of Random Variables in the Model	66
5.3	Analysis for Q_1 , that is, Priorities ≤ 2	80
5.4	Analysis for Q_2 , that is, Priorities ≥ 3	87
CHAPTER VI:	CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY	
6.1	Conclusions	89
6.2	Suggestions for Further Study	91
	BIBLIOGRAPHY	93
APPENDIX A:	SOME PROBABILITY DISTRIBUTIONS USED IN QUEUEING THEORY	97
APPENDIX B:	THE LAPLACE TRANSFORM OF A DISTRIBUTION FUNCTION	101
APPENDIX C:	THE DISTRIBUTION OF SOME RANDOM VARIABLES ENCOUNTERED IN BOTH PREEMPTIVE-RESUME AND HEAD-OF-LINE DISCIPLINES	103

LIST OF FIGURES

	<u>Page</u>
1. System state diagram	21
2. Program state diagram	22
3. The basic queueing model	26
4. Timing diagram for IBM 2314 disk storage	31
5. Queueing model of disk storage facility	32
6. Congestion in core storage system	33
7. The round-robin system	34
8. The FB_N model	40
9. Virtual addressing and address mapping tables	51
10. Allocation and movement of pages in memory	52
11. Scheduling logic of dispatcher in CP/67	59
12. Timing diagram of priority k level	64
13. Busy period for the k^{th} priority	64
14. Queueing model of dispatcher	65

CHAPTER I

INTRODUCTION

1.1 Studies and Analyses of Time-Sharing Systems.

With the advent of time-sharing systems providing 'simultaneous' access for a large number of users to a computer, several studies have been undertaken to decipher, evaluate, and improve their performance. Whereas other systems could be studied rather easily, these time-sharing systems have been more difficult because of their complexity. Some of the factors which account for this are the randomness of input to the system, the number of functions occurring at the same time, and the different programs simultaneously resident in core.

In every time-sharing system there is a set of algorithms for storage allocation, facilities assignment, and job scheduling. The performance of the system is dependent on the trade-off between these factors. Overhead in a time-sharing system may be defined as the slice of processing time the Supervisor spends on scheduling, allocating, buffering, and controlling terminal input/output (I/O) . The minimization of this overhead is the chief goal of system designers and analysts.

The studies on time-sharing systems have been of three types: simulation, measurement, and analytical approaches. Nielsen [37] carried out a simulation study on a time-sharing system for the IBM 360/67 which was, at that time, on order by the Stanford Computation Center. Through his study it was possible to locate some faults arising from the design of the system, and to determine the appropriate configuration needed

for this center. Some of the other simulation studies appearing in the literature are due to the Fine and McIsaac [26], Penny [39] and Scherr [44]. A measurement or statistics-gathering approach has been followed by Totschek [54] in which he found that the service time can be satisfactorily represented by the log-normal and hyperexponential distributions. Finally several analytical studies have been undertaken on these systems. Penny [39], who carried out a simulation and analytical study of a time-shared system, showed that an analytical study could be worthwhile, especially if made in conjunction with a simulation study.

An analytical study, he suggests, can :

- i) give valuable preliminary information so as to determine the scope of proposed systems;
- ii) reduce the quantity of work in, and the cost of a simulation study by providing estimates of various parameters on which extrapolation and intrapolation could be done; and,
- iii) suggest strategies, to be verified by simulation, which may increase the efficiency of the system.

Scherr [44] used all three approaches in his work and showed that his analytical results agreed quite closely with the simulation ones.

Parameters evaluated in the study of time-sharing systems can be classified according to whether they relate specifically to the user or the designer of the system. For example, the user is interested in knowing how long he will have to wait to get the results from his program. Thus for him, completion and waiting time (to be defined later) are the

most relevant parameters. The designer, on the other hand, wants to have some estimates on the utilization of the facilities of the system. These may be provided by examination of the busy periods and queue sizes at these facilities. In time-sharing systems, with programs not necessarily serviced to completion in one service, some guide is wanted as to what represents a suitable quantum of time to be given on each entry to the processing unit. Hence, these are some of the parameters which are investigated in the different studies: completion time, quantum size, waiting time, queue size, and length of busy period.

1.2 Type of Analysis Used in This Thesis.

This thesis contains an analytical study of the CP/67 Operating System implemented at the University of Alberta. Some known probability distributions are used for arrival rates and service times, where the distribution parameters have to be estimated. (It may be remarked that this study is complemented by a measurement approach by Gatha [27] and a simulation study by Stanger [47].)

Since various stages of an operating system involve handling queues of requests for users' programs, the theory of queues will be used to design an appropriate queueing model for the scheduling of these requests in this system.

This survey, in Chapter III, includes analyses of (i) disk storage by Abate et al. [1], (ii) core storage facility by Chang [8], and models of the dispatcher. These analyses, along with the discussion on paging and swapping in Chapter II, are given not only because they apply to the CP/67 Operating System, but also because it was felt

that, with their inclusion, this study would be seen in a better perspective. Two other analyses, not discussed in this thesis, but which the author feels contribute to the better understanding of time-shared systems, are those of Denning [22] on console behaviour and Coffman [13] on drum storage. Extensive use is made of the Laplace Transform of the distribution function for input and service rates. The two main advantages of using Laplace Transforms, are firstly, that, in the case of the service time, distributions other than the exponential can be handled, provided, of course, that their Laplace Transforms exist, and secondly, whilst the actual distributions are usually analytically intractable, their moments can then always be obtained (see Appendix B).

Time-sharing provides different rates of service to different users on the basis of their priorities. In accordance with this, preemptive priority queues are used in the model of the dispatcher. A more detailed discussion of priority queues and their development is given in Chapter III.

CHAPTER II

TIME-SHARING COMPUTER SYSTEMS

2.1 Evolution of Time-Sharing and Some Definitions.

Until just recently batch processing was the sole means of controlling the execution of jobs in the computing environment. The name derives from the fact that work is received and processed in batches. The computer is set up specifically for each separate job. Each unit of work is processed to completion and then the computer is set up for the next job. In this way there is considerable wastage of central processing unit (CPU) time while input/output (I/O) is being done. Time-sharing systems, in which several programs are executed "simultaneously", have been developed to improve systems performance by increasing CPU utilization time. In a time-sharing system which is chiefly serving on-line users, batch processing might be done in the background to utilize the system's resources more fully. One of the earliest references to the time-sharing environment as we know it today is Strachey's [50] paper of 1959. He proposed a system with a central processor and one or more I/O devices which function simultaneously by multiplexing the memory. The best known of the early systems is the Compatible Time-Sharing System (CTSS) developed at M.I.T. by Corbato et al. [19]. This was implemented on an IBM 7090 which was expanded with peripheral devices such as disk files, etc.

At present there are several time-sharing systems which may

be classified as either dedicated or general purpose. The first kind of dedicated system is that limited in scope to the solution of one class of problems. Examples of this kind are those systems used in the banking and air-line industries. The second kind of dedicated system is that with which users can communicate in only one language. Engineers and scientists may use such a system to obtain solutions to their calculations, e.g. the BASIC system.

The type of time-sharing with which we are concerned here is the general purpose one in which users can communicate in several languages. Such a system should also accommodate diverse devices such as typewriter terminals, display terminals, remote entry stations, etc. Although several computers available today are capable of supporting time-sharing systems, we shall limit ourselves to the IBM 360/67 system.

Since there are several interpretations of terms used in discussing time-sharing systems, it is appropriate here to give definitions of these terms as they shall be used henceforth. Ziegler [59] defines time-sharing as a communications-oriented method of using computers. It is a technique that permits concurrent utilization of the same installation by two or more persons working at remote devices capable of direct, on-line access to the data equipment. Each of the users of a time-sharing system must have the ability to inquire into and add to or alter information in files accessible to the data processing installation in random access fashion, on demand. The users are independent in that they may be working with different programs in different areas of study.

Hence, the main aim of a time-sharing system is to provide all users with better service by utilizing the physical resources as efficiently as possible. Physical resources, as mentioned here, refer to the CPU , I/O channels, memory storage, etc. Multiprogramming is one common strategy adopted to improve service. This is the simultaneous execution of two or more programs in the computer system. (Program here may be a user's program, a segment of a user's program, or the supervisory program.) For example, if a program currently being processed in the CPU requires an I/O operation, another program may enter the CPU and the I/O processor handles the I/O request. The switching of programs from processor to CPU and vice versa involves some overhead in response time. This is an inherent evil of the time-sharing system and much consideration is spent on minimizing this overhead. Another strategy is that of multiprocessing where there are several CPU's in the system. This makes for greater efficiency in executing programs.

Great use is being made of the concept of virtual computers or machines which embody most of the techniques of multiprogramming and multiprocessing, at least on a virtual basis. As the virtual machine concept is used in the CP/CMS system, a description is given of this concept in the next section.

2.2 Virtual Machines.

A time-sharing system, as defined in the previous section, provides a set of software facilities through which users share machine resources. The extent of this time-sharing facility is largely determined by the available software. A computing system utilizing virtual machines

could be developed to provide this facility. Accordingly a description of the structure and use of such a system of virtual machines is presented here.

In batch processing systems some programs may be written in which the user has control over the allocation of resources e.g. CPU, memory registers, etc., during the execution of the program. He may also have almost complete control over the real time sequence of events during the execution of his program. Interrupts provide the exception. In time-sharing computer systems the user also controls the real time sequence of events in his program. However, he has little explicit control over the allocation of physical resources among different programs in the system. This allocation of resources is performed dynamically by the system. Thus it is convenient to store information and programs in a hardware-independent manner during execution. The concept of virtual machines was developed as a result of this realization (see Wagner [55]).

The virtual machine, simulated by software, is the one envisaged by the user in writing his programs. It is assumed that a time-shared computer system, e.g. CP/67, can accommodate several hardware-independent virtual machines or virtual computers. Though different from a real computer, the virtual one is programmed as if its resources were dedicated to the users. The run-time representation of instructions in virtual machines is referred to as the virtual machine language. This cannot and does not refer to the physical storage registers but rather utilizes virtual addressing. The correspondence between virtual and physical addresses is kept for each user in a set of address mapping tables e.g. SEGTABLE in CP/67. These tables and the correspondence between

them are updated whenever a block (segment or page) of information is moved.

There is considerable freedom in designing virtual address space. It should be related to the physical address space so that mapping virtual addresses to physical ones by the address mapping table can be rapidly done. Also the virtual address space should be designed for the convenience of the user as far as possible with the stated constraints. One virtual address may, at different times during computation, correspond to different registers in real memory. More important, virtual addressing permits addresses of different virtual machines to denote the same physical address. Therefore information, common to several users as for example system routines, can be referenced by them at the same time.

In the description of CP/67 (Section 4.2.3), a diagram is provided illustrating virtual addressing more clearly.

2.3 The Operating System.

In batch processing computer installations where each job must follow the same flow through the system, an operator could efficiently manage the allocation of resources. The development of large general purpose time-sharing systems has made this impossible. We are concerned here with the software component and, as a result, discussion of the hardware is limited to the ways in which it affects the implementation of the specified software.

The operating system software must be sufficiently general to provide a variety of applications on a wide range of hardware configurations.

It consists of two programs, a control program and processing programs. The processing programs consist of language translators (such as FORTRAN compilers) and service programs (such as the LINKAGE EDITOR). The control program is referred to by some writers as the SUPERVISOR or EXECUTIVE. It has three main functions:

- i) scheduling of jobs on a continuous flow basis;
- ii) supervision, on a sequential or priority basis, of each unit of work to be done; and
- iii) storing, retrieving, and maintaining all data irrespective of the way in which it is stored and organized.

This then is the aspect of the time-sharing environment in which we are primarily interested.

2.4 Memory.

Memory in a time-sharing system is a hierarchy of storage devices containing both users' programs and the operating system software. There are two levels of memory, primary memory and secondary memory. Core storage is usually used to provide the primary memory of today's computer systems. In different papers it may be referred to as main storage, high speed access memory or, simply, core.

Primary memory, that which the CPU is capable of addressing directly, is a set of fast access blocks of information pages. In it are all the active portions of users' programs and the operating system software. In some time-sharing systems all the system software is stored

in core, this, however, limits the amount of core available to users' programs. For communicating with primary memory, wherein several pages of data from several different programs are resident, addressing tables and I/O channels must be maintained. In this way, the pages can be referenced simultaneously, or at least, in immediate succession. The concepts of paging, the division of programs into blocks, and swapping, the transfer of data in and out of main storage, are discussed in some detail in the next section. Secondary memory, often referred to as auxiliary memory or backup storage, provides storage facilities for the remainder of the users' and system programs and data files. The secondary memory is not directly accessible by the CPU and hence, I/O commands and the respective I/O channels are necessary for information transfer from secondary to primary memory and vice versa.

Hence, the two main differences between primary and secondary memories are the access time and the size. Access is defined here as the time required to get a page of information from memory given that an I/O channel is free. Since primary memory contains the active parts of programs, it is essential that its access time be considerably less than for secondary memory. Both disk and drum storage are used for secondary storage. The access time for drum storage is shorter than for most disk files for two reasons; first, each track or channel of the drum has its own head, and second, the positioning time for the head at the relevant data block is faster. The average access time are 8.6 m-seconds for the 2301 drum and 87.5 m-seconds for the 2314 disk [29]. Secondary memory, on the other hand, is bigger than primary storage. The cost of maintaining the latter dictates this; otherwise there would

be no need for auxiliary storage, nor, indeed, for paging. In view of the great differences between primary and secondary memories as far as speed and costs are concerned, much research has been done towards finding efficient paging and swapping algorithms. Some of the strategies suggested are presented in section 2.4.2.

2.4.1 Paging.

Paging is the fragmentation of users' programs and, in some cases, systems programs into a number of blocks of information or pages. This causes considerable reduction in system overhead incurred by physically moving information into core. With programs stored in pages, there is less likelihood that several pages of unwanted information would be brought into core. The size of pages may vary from one installation to another; similarly the amount of core storage available may depend on the need for it, and cost involved in providing it. Associated with each page is a bit indicating whether or not the page has been written in since it was last copied. A record of the usage of pages is kept in the page table [23].

To facilitate easy page management, several pages may be grouped together to form a segment. This division of programs into pages and segments is usually invisible to the user. In some instances the user, or preferably in this case, the programmer can indicate the divisions he desires. He is then able to tackle portions of his program at a time rather than all of it at once [41]. Greater efficiency should result.

Pages of an active program which follow each other in sequence do not have to occupy contiguous locations in memory, and, in fact, they seldom do. This can be readily understood since they might have been brought in at different times and thus would have to occupy the available empty spaces (see Figure 10). Page tables and segment tables help in tracking and chaining these pages randomly scattered throughout the computer memory. The segment tables contain the physical or real address of the segment and the location of the tables in which its pages are defined. The page table contains the physical addresses of the pages and information relating to their state. The addressing is identical to "virtual addressing" (Section 2.2.).

The very nature of time-sharing systems makes it necessary to have some protection for users' programs. Since supervisory and library programs have to be protected from any modification whilst being used, and confidential sections of users' programs must have restricted access, such pages may be locked so that they can be brought into core and referenced, but their contents cannot be displayed. Although the storing of the address mapping tables in core lessens the available space, paging and segmentation do improve the execution time for programs.

2.4.2 Swapping Algorithms.

The swapping algorithm, in conjunction with the paging algorithm, makes space available, say a vacant page, in memory to accommodate a required page. There are either vacant pages in main memory, or the algorithm must determine which pages to remove. As defined by Denning [21], page swapping is carried out in two stages:

- i) paging-in: locate the required page in auxiliary memory, load it into main memory, turn the "in-core" bit of the appropriate page table entry ON.
- ii) paging-out: remove some page from main memory, turn the "in-core" bit of the appropriate page table entry OFF.

Page traffic, the number of pages per unit time being moved between memories, may be considered a measure of performance for a swapping algorithm. The minimization of this traffic seems a logical strategy to adopt, since

- i) any increase in data traffic between memories brings an increase in the overhead; and
- ii) the transfer time of pages between memories is large compared with the processing time, hence, too much data movement could result in the congestion of the I/O channels between memories.

Although there have been some attempts at anticipatory page-loading, (see Ramamoorthy [42]), we shall, as is generally done, assume that paging is on demand. Insufficient prior knowledge of the ordering and running time of programs rules out the first approach. The major consideration in memory management is deciding which pages to remove from main memory. If the page with the least likelihood of being used again is removed, the best choice has been made. Alternatively, if the page removed is immediately requested, then that would have been the

worst possible selection. Some of the suggested strategies for page-swapping are now examined:

RANDOM SELECTION: Whenever a fresh page of memory is needed, a page to be replaced is selected at random. The implementation of this strategy is apparently very simple but frequently seems to remove useful pages from core.

LEAST RECENTLY USED (LRU) SELECTION: If a page is to be paged-out, the one unreferenced for the longest time should be removed. This strategy can be implemented with relative ease because (i) each page-table entry contains a bit determining whether the page has been referenced, and (ii) the page-tables can be checked at frequent intervals and the usage recorded and updated. However, there may be overloading of the I/O channels when several programs compete for main memory.

FIRST-IN-FIRST-OUT (FIFO) SELECTION: Whenever a fresh page of memory is needed, the page least recently brought in is removed. The implementation here is also quite simple. The pages of main memory are stored in a cyclic order; this is based on the assumption that programs tend to follow sequences of instruction, i.e. references in the immediate future will be close to present ones. Pages residing in main memory the longest are the least likely to be used again and thus cyclic ordering is justified. (This is identical to the cyclic selection used by Baylis et al [3] in their study on the Atlas Computer.) It is unlikely that the basic assumption of this strategy is always valid.

With identical job streams, the following results were obtained [3]:

- i) the page traffic with the FIFO selection was 10% higher than for the LRU;
- ii) random selection increased the traffic by a factor of over two; and
- iii) a reduction in the size of a page and an increase of pages simultaneously in core result in greater efficiency.

Fine et al. [25] have seriously questioned whether paging is worthwhile. In discussing their work, Denning [21] wrote; "We cannot agree more with their data, nor agree less with their conclusion". Belady [4], whose Optimal Replacement Algorithm requires a prior knowledge of the entire sequence in which pages would be used, defines an 'ideal' algorithm as having the simplicity of Random or FIFO as well as some historical data on its usage.

2.5 The Supervisor.

The Operating System in a multiaccess and/or multiprogramming time-sharing system provides apparent simultaneous availability of its resources to the different users. The Supervisor of this Operating System must efficiently control and allocate all the resources of the system to the different users. Its function, then, is to provide an interface between the software and the hardware. Any conflicts in requests for the resources are resolved by the Supervisor according to some priority discipline. The Supervisor maintains complete control of

the system for the duration of the processing, administering all interruptions. One important characteristic of the Supervisor is that it cannot be directly referenced by another program or 'virtual machine'.

While there are several good descriptions of the Supervisor in a time-sharing system, the one presented here is, in essence, the one stated by Wood [58]. The general approach and well organized layout of his presentation precipitated this choice. The major components of a Supervisor are:

- i) The Interrupt Processor;
- ii) The I/O Processor;
- iii) The Timer Administrator;
- iv) The Program Storage Administrator;
- v) The Facilities Administrator;
- vi) The Program Administrator.

2.5.1 The Interrupt Processor.

The Interrupt Processor, the heart of the Supervisor, is made up of two parts, the Hardware Interface and the Initiation Routine. The former accepts and responds to all physical interrupts generated by hardware. These may be classified as follows:

I/O Interrputs caused by the completion, successful or not, of an I/O operation. These are generated when a device has completed a requested operation and is free to handle another request or command.

Timer Interrupts caused by the overflow of the hardware clock. This is used by the Supervisor to prevent any program from exceeding its allotted CPU quantum and dominating the system. This and operator-generated interrupts are usually classified together as External Interrupts.

Program Interrupts caused by faulty execution of certain machine codes. These include overflow and underflow in arithmetic operations and attempts by one program to reference another directly.

Hardware Interrupts or machine interrupts caused by malfunctioning of the hardware.

Supervisor Call Interrupts (SVC) caused by a special instruction in the program. This differs from a Program Interrupt in that use is made of the automatic interrupt linkage. It is used most frequently to switch from problem state to supervisor state.

2.5.2 The I/O Processor

Since requests for I/O activities may be generated faster than they can be handled, those for a pending operation are placed in Request Queues for the particular peripheral devices. An entry in this Request Queue describes completely the nature of the operation to be performed, e.g. the requested operation and the address of the program requesting the operation.

With I/O processing, the flow of data between the CPU and main memory storage is the prime concern. Moberg [36] suggested the following rules as guides in their processing:

- i) in determining what action to follow, I/O operations should have highest priority;
- ii) job priority should not be applied in I/O processing;
and
- iii) swapping programs with outstanding I/O request would obviously be uneconomical.

The I/O interrupt is responsible for validating that the request information has been transmitted.

2.5.3 Timer Administrator.

In a time-sharing environment the order in which operations are started may be, and quite often is, substantially different from that in which they are completed. There are some occasions when it is necessary to switch to another operation rather than wait a relatively long time for the completion of a particular operation. In a virtual machine environment there are real and virtual clocks in the system which are maintained by the Timer Administrator. These may be used to keep an account of the time spent in the system and the processing time given each program so that costs may be calculated. It is also important in studies such as this one to have checks on the times spent in the queues to the various resources.

2.5.4 Program Storage Administrator.

During the course of the execution of a program, the program is resident in the computer's memory or Program Storage as referred to by Wood [58]. Associated with each program in the Program Storage area are various descriptors. These define the files attached to the program, the address mapping tables such as page tables and swap tables, and the data set assigned to the program.

The Supervisor controls and allocates programs, both users' and system programs. They are prevented not only from dominating the CPU but also from overloading main memory. The Program Storage Administrator, making use of the Paging and Swapping algorithms, allocates memory storage to all programs. It oversees the transfer of data from auxiliary memory to main memory and then from main memory to the CPU.

2.5.5 Facilities Administrator.

The unavailability of facilities is the main cause of congestion in time-sharing systems. These facilities include registers, tape files, buffers, channels, logic and control circuits, printers and card readers. The problem is made more complex by the fact that these facilities generally operate asynchronously. For example, the availability of an I/O channel might not coincide with the availability of the required buffer. The Facilities Administrator looks after the task of satisfying requests for all the facilities. It prevents the simultaneous assignment of one of the computer facilities to several tasks, or of a given task to several of the facilities.

2.5.6 Program Administrator.

The overall status of the operating system is determined by three mutually exclusive program states, Wait, Supervisor and Program. The "wait state" is entered whenever there are no programs or requests to process. From this, the "supervisor state" is entered when there is some indication that an operation has been completed or must be initiated. All interrupts are handled, I/O initiated and response made to all Supervisor calls when the system is in this state. In the "program state", programs are processed by the CPU. The ways in which the system changes its status are clearly seen in Figure 1.

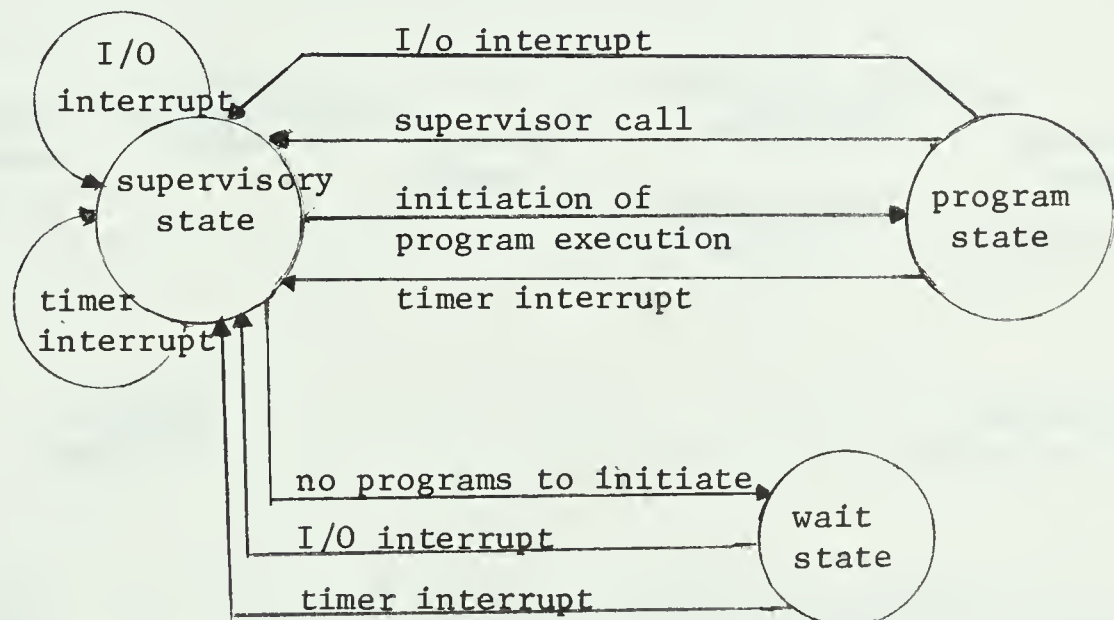


Figure 1. System state diagram

The program itself can be in one of several states at any instant during the processing. These states are classified here as follows: Program Definition, Program, Execution, Service, Service

Pending and Service Delay states. The Program Definition state is that in which the program is being assigned to the various facilities and is being provided with the required data sets. When these facilities have been assigned, the program now moves to the Program state where it vies for a place in the queue for CPU service. When this place is obtained, the Execution state is entered and the program is eligible for service from the CPU. In the Service state, reached after an SVC interrupt, the program awaits service. The Service Pending and Service Delay states are entered according to whether service is completed or cannot be started. The Program Administrator controls the flow between these states and determines the priorities among programs (see Figure 2).

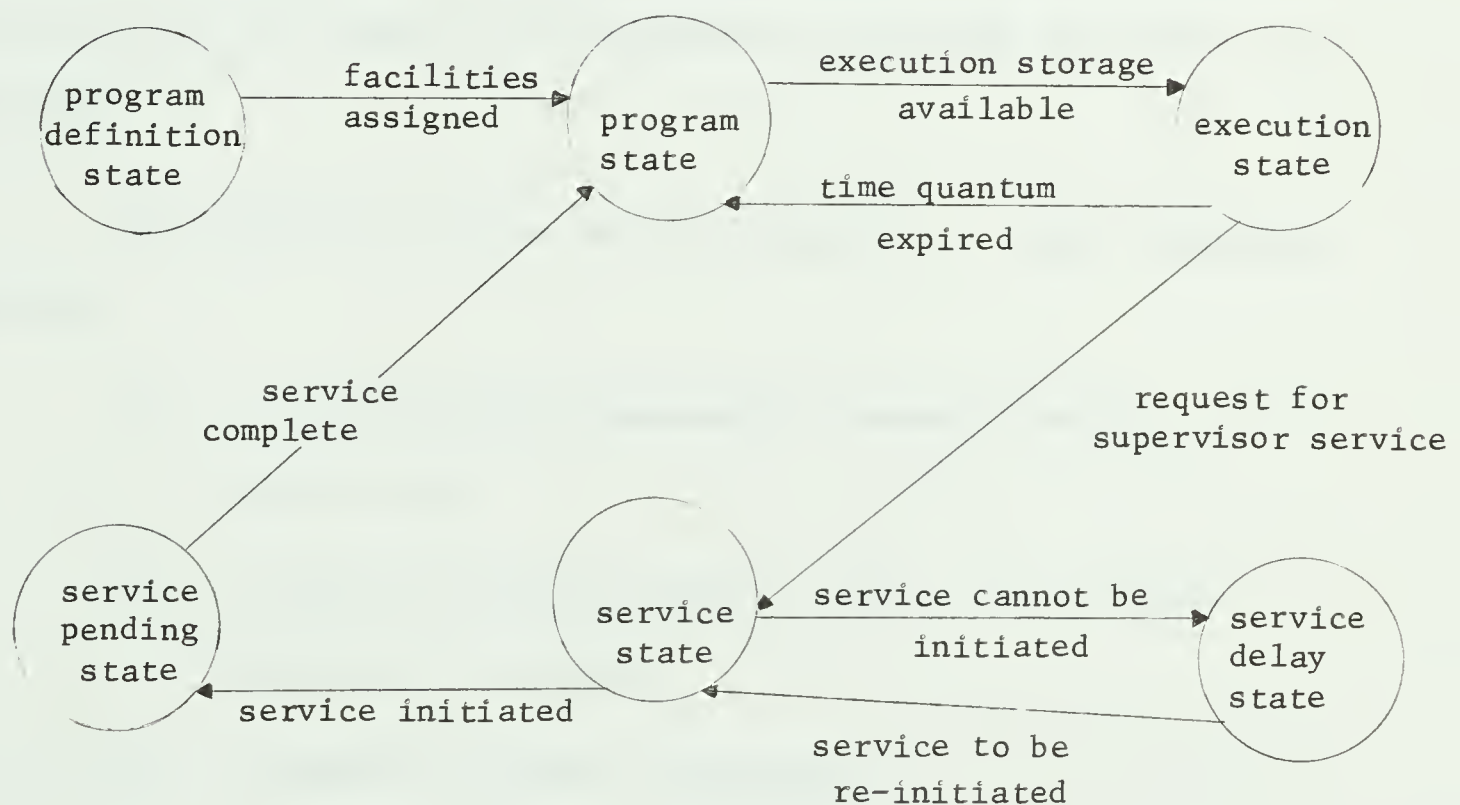


Figure 2. Program state diagram

2.5.7 The Scheduler/Dispatcher.

The Program Administrator in control of the movement of programs from one state to another makes use of a scheduler which allocates the CPU time. The Scheduling algorithm may be designed to maximize system's efficiency by reducing overhead caused by swapping. Alternatively, user efficiency might be the aim of the algorithm. A good scheduling algorithm should recognize both goals, and should attempt to provide satisfactory service for the user at an efficient operating level. All active programs, i.e. those in the Execution state, are visited by the scheduler at appropriate intervals and are given a slice of CPU time. These appropriate intervals may be constant, but, in most cases, they are dynamically determined from consideration of the state of the system, i.e. the number of programs and their priorities.

The following are some of the essentials of a good scheduling algorithm:

- i) it should provide "reasonable" response for the on-line user;
- ii) it should, wherever possible, reflect the different priorities of programs;
- iii) it should be simple to implement;
- iv) it should be alterable with minimum effort. (This is very important since programmers differ substantially in what they consider as efficient algorithms.)

The simplest scheduling algorithm is the round-robin (RR) .

In an RR system, each job or service request is serviced for a quantum or slice of processing time (up to a maximum of q seconds); if service is completed during the quantum it leaves the system (i.e. queue and CPU), otherwise the job is cycled back to the end of a single queue to await another quantum of service. The parameters for the RR algorithm are the quantum size and the cycle time. If the quantum is constant, then the cycle time is the product of the quantum and the number of users present. The length of a quantum also affects the overhead in a swapping environment since the shorter the quantum, the more swaps are necessary. The quantum may, however, be of variable length and with varying number of programs only a probabilistic estimate of the cycle time may be obtainable. Pyke [41] has suggested that the cycle time should not be longer than the human reaction time of about one-tenth of a second. A program requiring more than one quantum of service will have to wait for several cycles.

In most systems some degree of priority scheduling is employed. The priority levels may be determined before run-time; for example, with CTSS where the system estimates the processing time required from the length of the program, or at the University of California at Santa Barbara [15], where users are requested to declare whether their programs are interactive or batch. Longer or compute-bound programs are placed in lower priority levels where they are given longer quanta at less frequent intervals. Shorter or interactive programs, on the other hand, are processed more frequently but for shorter quanta. If a program has not been completely processed at the end of its quantum,

it is fed back to await further processing. In multilevel priority scheduling, priority within each level is generally in the order of arrival. The RR discipline consists of only one queue for service in which service may be in order of arrival or by priorities.

These are two approaches for scheduling algorithms. Analytical studies of these and other strategies are given in the next chapter.

CHAPTER III

A SURVEY OF QUEUEING THEORY APPLICATIONS IN TIME-SHARING COMPUTER SYSTEMS

3.1 Introduction to Queueing Theory in Time-Sharing Systems.

Queueing theory is concerned mainly with predicting various characteristics of waiting lines, or queues such as the average length of the line and the average waiting time in the queue. The basic queueing model consists of four parts: the input source, a system of queues, the service discipline and the actual service facility (see Figure 3).

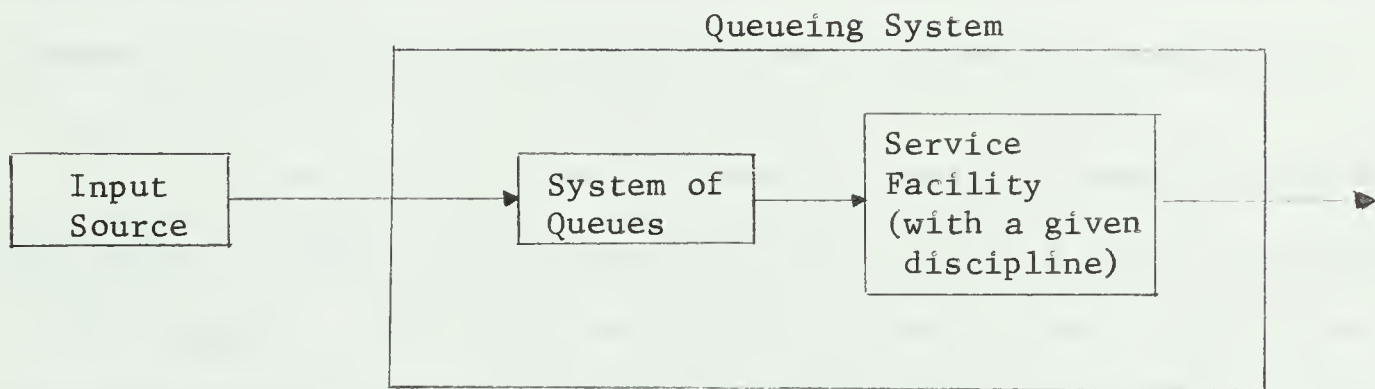


Figure 3. The basic queueing model

The input process has two main characteristics, its size and the arrival pattern. The arrivals can be from an infinite population or a finite population. The population is assumed to be infinite when there are a relatively large number of units entering the system. In

the case of a finite population, the analysis is more difficult since the number of units in the system determines, to some extent, the number which can be providing input to it. For the arrival pattern, a statistical distribution is selected according to the empirical data collected. Some of the more common distributions are the Poisson and Erlang distributions (see Appendix A). It is essential that any irregularity in the behaviour of an entering unit be noted e.g. balking (i.e. leaving the system because the queue was too long) or deflecting (i.e. leaving when the waiting time becomes excessive).

Queues may be finite or infinite in length and in the system there could be more than one queue. Within each queue there could be different methods of selecting the next unit for service, as for example, random selection, priority procedures or first-come-first-service (FCFS). A model in which there are several queues is given in Section 3.2.5.

In most applications of queueing theory in computer systems the units or tasks have different priorities and this must be reflected in the analysis. Such cases necessitate priority service disciplines, in which case the service may be preemptive or non-preemptive. A look at some priority disciplines with some relevant history is given in Section 3.1.1. Non-priority models have also been studied and in them the tasks are served to completion or for a given slice of time. Probability distributions are assumed for the service time.

The service facility could consist of servers in series, or in parallel, or simply a single server. The seek phase of the model of the disk storage facility in Fig. 5 provides an example of servers

in parallel. Multiprocessing, also, is an example of servers in parallel. The model of the disk storage facility is one with queues or servers in series. The CPU, I/O processor, etc. are examples of single servers.

Queueing theory, then, is quite suitable for studies of time-sharing systems wherein there are several queues of requests for their facilities. Some of these studies will be given after a brief outline of priority queues.

3.1.1 Aspects of Priority Queues.

Priority queues can be classified as being non-preemptive or preemptive. In the non-preemptive or head-of-the-line discipline, when the task being served is completed, the highest priority one is given service.

The preemptive discipline, in which current service is interrupted in favour of a higher priority unit, can further be divided into two categories: preemptive-resume and preemptive-repeat. In the former class, the preempted unit resumes service from the point at which it was interrupted. With the latter category, service given to the unit before preemption is wasted and it must be given all of its quantum again.

The first published work on priority queues was by Cobham [11] who obtained the expected waiting time for a queue with Poisson arrivals and general independent service times. Phipps [40], and Kesten and Runnenberg [30] obtained the Laplace transform of the waiting-time distribution. Preemptive priority disciplines were introduced by Stephan [49], and White and Christie [57].

Miller [35] gave detailed consideration to the preemptive-resume discipline. He showed that the distributions of some parameters such as the busy period are the same for both the head-of-the-line and preemptive-resume rules. Gaver [28] obtained results for several parameters using the three disciplines described above.

Chang [9] and Schrage [45] have both used priority models for the study of time-shared systems. Schrage developed two models: one, in which tasks are feedback to lower priority levels, and the other, where the order of service is dependent on the processing time required by the task. Chang extended Takacs' [51] feedback model to include priority tasks.

3.2 Survey of Some Analytic Studies of Computer Systems.

Most of the analytic studies of computer systems have concentrated on the dispatcher. This trend can well be appreciated when the role of the CPU, and therefore, the dispatcher is examined. The CPU is the center of activity in a computer system. However, in an efficient system, memory utilization, channel utilization and swap time play, an important part in the working of the CPU. We have, therefore, included in our survey two studies, one on a disk storage facility, and the other on congestion in core storage.

Studies on the dispatcher by Shemer [46], Schrage [45], and Coffman [12] are summarised. Shemer and Schrage, by assuming swap time to be zero, have obtained results which, as Kleinrock [31] observed, can only be considered as upper limits on the service attained. Coffman,

whilst not making direct allowance for swapping, proposed two approaches which would reduce it (see Section 3.2.5).

3.2.1 Queueing Analysis of Disk Storage Facility.

Storage devices, such as drum(s) and disks, play an important role in determining the level of performance of time-sharing systems. Hence, we shall consider one of the queueing analyses for such a device, namely Abate et al.'s [1] paper on the performance of an IBM 2314 disk storage. Disk storage is widely used in the computing field and its use in the CP/CMS system is of particular interest to us.

The IBM 2314 disk storage device consists of a control unit and eight on-line storage modules, each containing a disk pack and its own access arm [29]. An I/O channel is used in connecting this device to the CPU. A typical read operation is divided into two phases, the seek phase, and the access and transmission phase. During the seek phase the arm is positioned at the desired cylinder and in the second phase, as the disk pack rotates, records containing the data reach the "read" head (rotational delay) and then are transmitted via the I/O channel. While the other access arms may be positioned during the transmission of data, transmission can only be initiated when the channel is free, and hence it can only be from one disk at a time. This study obtains the 'file system response time' distribution using the Laplace transform (see Fig. 4).

The queueing model, considered by Abate et al. [1], consists of two queues in series or tandem. The first queue represents the seek phase and the second one describes the I/O channel 'service' time

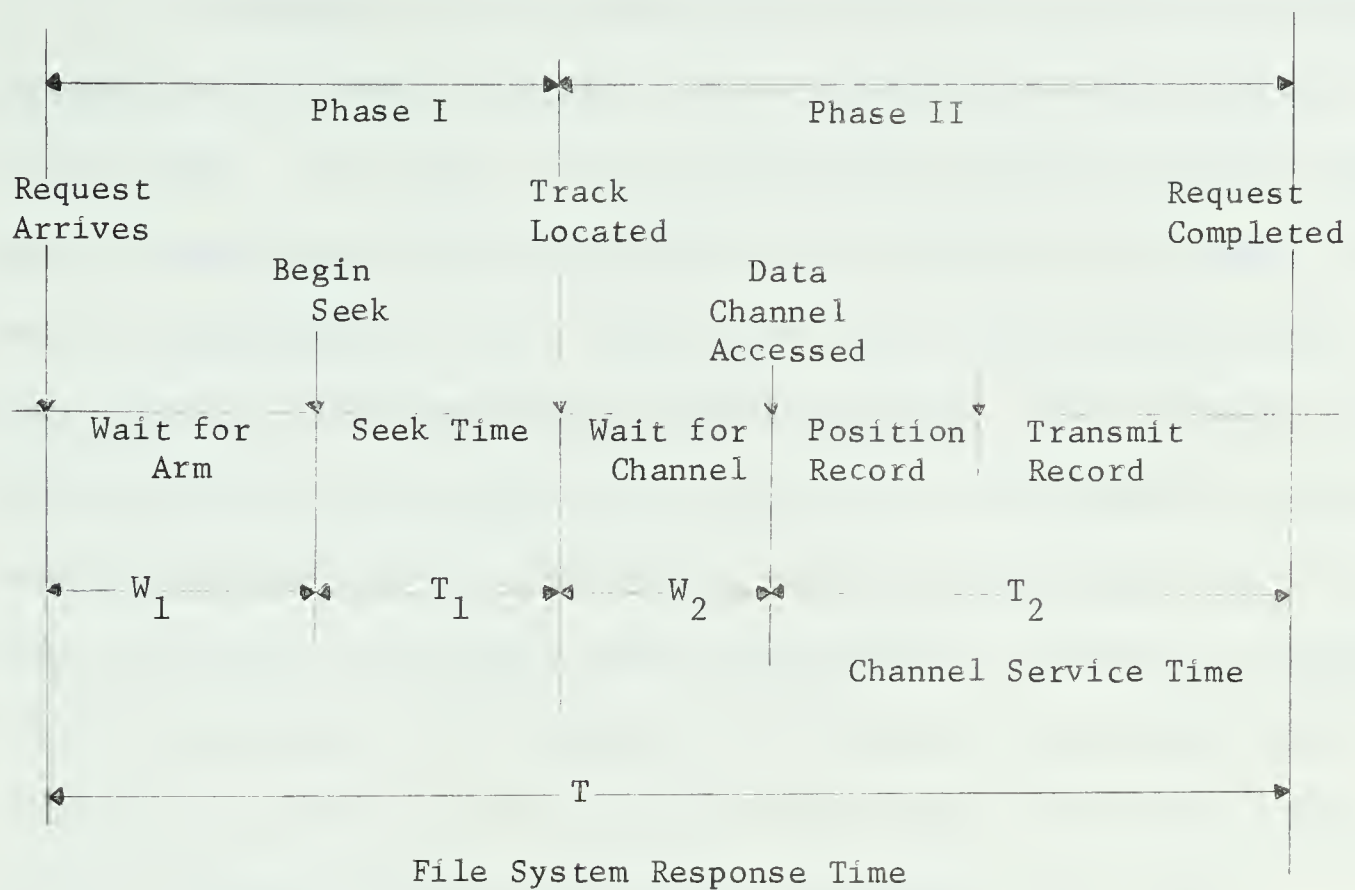


Figure 4. Timing diagram for IBM 2314 disk storage

(i.e. the time a request remains in the I/O channel). The file system response time, T , is given as the sum of four random variables (see Fig. 4);

$$T = W_1 + T_1 + W_2 + T_2 .$$

The assumptions necessary to solve the model analytically are:

- i) requests to the system arrive in a Poisson manner and then are uniformly distributed to the eight disks (i.e. each arm gets $1/8$ of the requests);
- ii) the sources of requests is infinite and the order of service is FCFS ; and,
- iii) the seek queue and the channel queue are handled as independent queues. In this way the input to the channel queue is a Poisson process.

In Figure 5, the queueing model illustrating the interaction between the two queues is given. There is one obvious shortcoming in the model. The access arm, on reaching the specified cylinder, must await transmission of the data before it can initiate a new seek. Thus, with a probability of $1/8$, the access arm, whilst occupied with one request, will be required to initiate another. This introduces an additional delay not accounted for in the model. This delay is almost unnoticeable with low arm utilization but for a high input rate it is the wait time in the channel queue which greatly increases the response of the file system. The treatment of the input to the channel queue is shown to be correct for the case of Poisson input to the first queue [43]. In general, however, this third assumption is not valid.

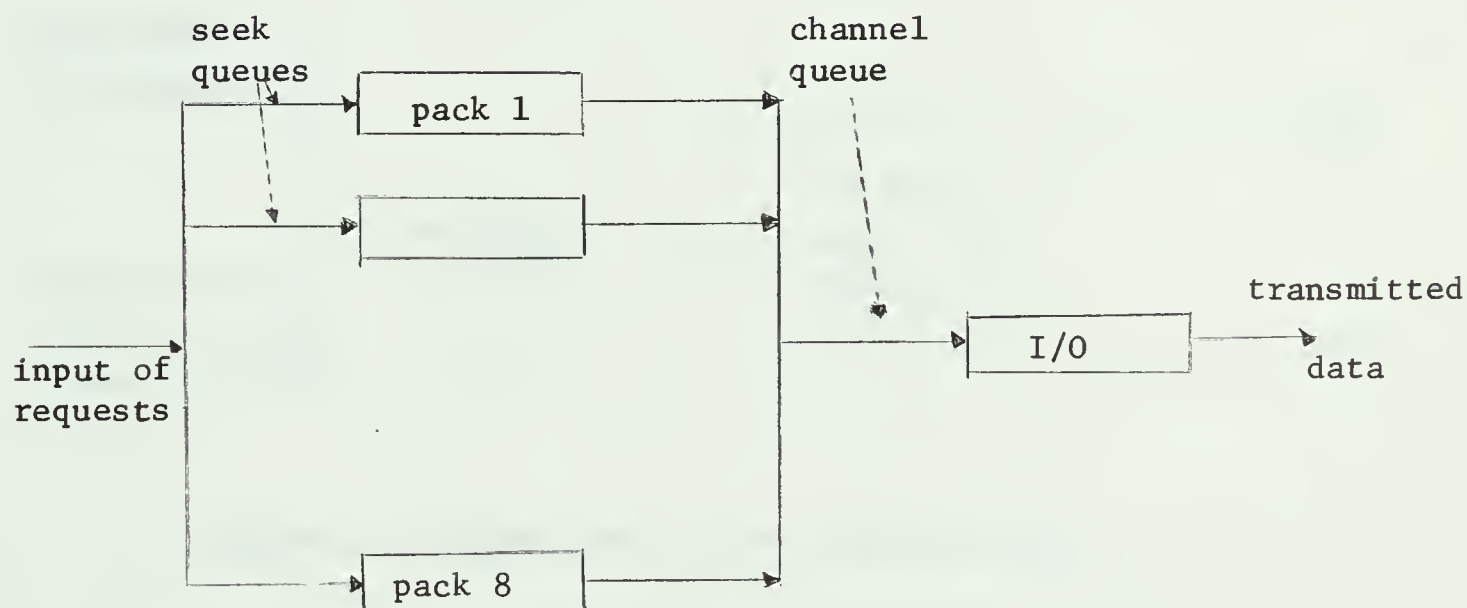


Figure 5. Queueing model of disk storage facility

For their model, Abate et al. [1] obtained expressions for the distribution of the times spent in the seek phase, the channel phase, and the total response time. The density function of the last was derived

from the convolution of the density function of the other two. Since the expressions are rather complicated, they shall not be repeated here.

3.2.2 Congestion Analysis of a Computer Core Storage System.

Another example of queueing analysis is Chang's [8] study of congestion in the storage control unit (SCU). As defined by Chang, the SCU controls and coordinates the transfer of data from main storage to either the I/O channels or the CPU. The SCU is used in installations where the channel functions are controlled by the I/O processor and not directly by the CPU. The analysis assumes two-way interleaving of storage requests by which the storage is divided into two sections, the even address and the odd address section (see Fig. 6).

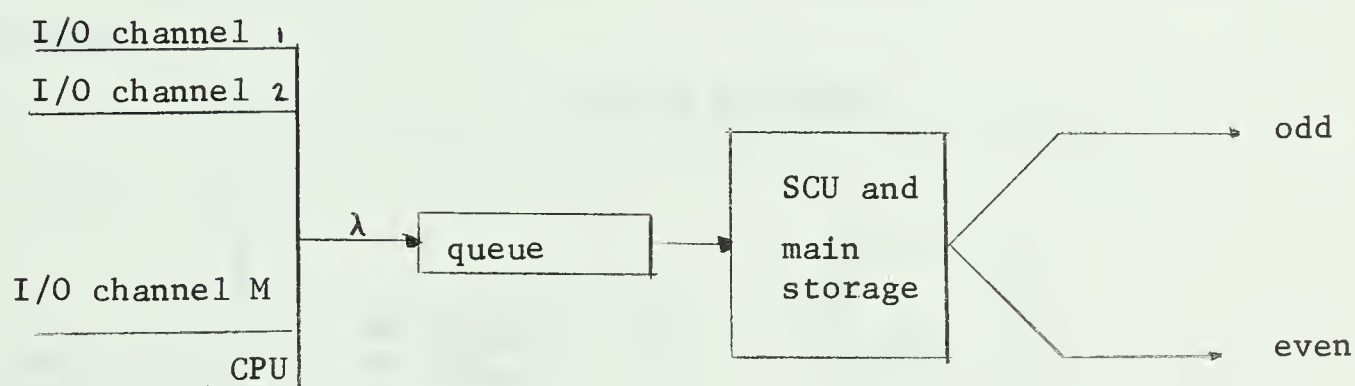


Figure 6. Congestion in core storage system

In his model, Chang treats the storage requests as if they were all of the same priority but with the appropriate modifications, different priority requests could be accommodated. Storage requests are assumed to arrive in a Poisson process from an infinite source. The SCU and main storage are combined to form a single-server facility

whose service time is the same as that of a storage request in main storage. Requests are serviced either singly or in batches of two, (an odd and even address request). The Laplace Transform is used to obtain expressions for the average number of storage requests and their waiting-time distribution.

3.2.3 Shemer's Mathematical Consideration of Time-Sharing Scheduling Algorithms.

Shemer [46], outlines the mathematical techniques for evaluating a time-sharing system. Although expressions have been obtained for only the expected response time, his study provides a fundamental basis for the analysis of RR and priority scheduling algorithms. Response time is defined as the time for the system to react to a user's request for service.

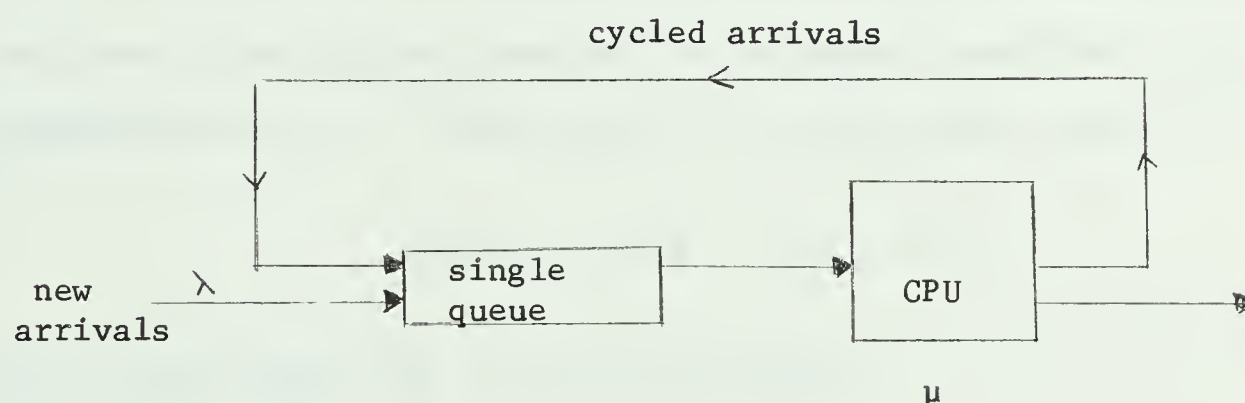


Figure 7. The round-robin system

In the RR model, he assumes a Poisson input with arrival rate λ and exponential servicing with parameter μ . The steady-state probability of n users concurrently making requests, as shown in Saaty [43], is

$$P_n = \rho^n (1-\rho) \quad \text{with} \quad \rho < 1, \quad \text{where} \quad \rho = \frac{\lambda}{\mu}.$$

Some authors refer to ρ as the utilization factor. The arrival and service rates are assumed to be independent and the queue size to be infinite. The expected number of users demanding service (either in queue or being serviced) is $E(n)$ where

$$\begin{aligned} E(n) &= \sum_{n=0}^{\infty} n P_n \\ &= \frac{\rho}{1-\rho}. \end{aligned}$$

Constant time quantum with RR service. Each user's request receives the same time quantum unless the requested service is finished before the end of the given quantum. The order of service is FCFS with return to the end of the queue if the request has not completely been serviced in the allotted quantum. For a request requiring total processing time, t , there will be N time quanta, where

$$(N-1)q < t \leq Nq, \quad \text{and} \quad N \geq 1$$

for N an integer and q the time quantum.

The expected response time, $E(R_N)$, is then

$$E(R_N) = W_N + t,$$

where W_N is the average waiting time accumulated in N allocations. $E(R_N)$ can be obtained by calculating τ_i , the conditional expected wait time between the completion of the $(i-1)$ -th cycle and the start of the i^{th} , given that the job has undergone $(i-1)$ time quanta.

We then have,

$$W_N = \sum_{i=1}^N \tau_i .$$

Shemer [46] further develops expressions for $\tau_1, \tau_2, \dots, \tau_i \dots$ to show that if parameters λ , μ , and q are known, the expected response time, $E(R_N)$, can easily be obtained for a given total, t , of processing time.

Variable time quantum with RR service. A request with total processing time, t , will be allotted N' slices, where

$$\sum_{i=1}^{N'-1} q_i < t \leq \sum_{i=1}^{N'} q_i, \quad N' \geq 2, \quad N' = 1 \text{ if } t \leq q .$$

There q_k is the k^{th} time allocation for a specific request. The quantum for each request is a function of the processing time already received. As before, the expected response time can be calculated from the average waiting time and the total processing time, t .

Other cyclic (RR) service rules can be investigated such as time quanta inversely proportional to the queue size, and the termination of quanta whenever an I/O request is encountered. The latter would tend to increase the number of preemptions necessary.

Priority discipline. In the priority discipline each request, on entry to the system is assigned a certain priority and receives service only when there are no requests from a higher priority class. Subsequently, after each time slice (in the non-preemptive case) the priority is diminished by one. With exponential inter-arrival and service times,

the expected waiting time between requesting service and the service initiation for a request r_j from the j^{th} priority class, is denoted by $E(W_j)$.

$$E(W_j) = E(T_o) + \sum_{k=1}^j E(T_k) + \sum_{k=1}^{j-1} E(T'_k),$$

where

$E(T_o)$ is the average time to complete the current time quantum when r_j arrives.

$E(T_k)$ is the average time necessary to service queued requests with priority higher than r_j .

$E(T'_k)$ is the average time to service those requests joining the queue after r_j , but having higher priority.

Given that r_j receives N quanta, its expected response time, $E_j(R_N)$, is

$$E_j(R_N) = \sum_{k=j}^{j+N-1} E(W_k) + t$$

This average response time is derived from the initial priority class of the request and its subsequent passage through the different priority levels.

Shemer [46] also discusses why he makes the assumption of zero overhead for swapping.

3.2.4 The SRPT and SPT Disciplines by Schrage.

With the shortest-remaining-processing-time (SRPT) discipline,

an arriving request will preempt the request currently being processed, if the processing time of the arriving request is less than the time for completion of the current request. When a request has been serviced to completion, the one with the least remaining processing time (least processing time for those yet to receive their first quantum) is selected. The SRPT discipline becomes shortest-processing-time (SPT) if there are no preemptions and the tasks (requests) are serviced to completion. Schrage [45] listed the following conditions under which the SRPT and SPT disciplines have optimal characteristics:

i) if the processing times are known at arrival, but not earlier, then for an $M/G/1$ queue (Poisson input, service distribution $B(t)$, where $B(t)$ is an arbitrary distribution for finite t , and one server) the expected time spent in the system is a minimum under SPT.

ii) if processing times are known on arrival, but are not affected by the number of preemptions while in the system, then for a system whose arrival and service rates can be represented by arbitrary distributions, $A(t)$ and $C(t)$, the average time spent in the system is a minimum under the SRPT discipline for the first condition, the processing times are assumed to be independent while no such assumption is necessary for the second. In the SRPT discipline, tasks are processed in order of earliest completion, and hence the mean number of jobs in the system at any one time is minimized. The situation might arise when the job with the shortest (remaining) processing time is fractionally less than the job currently being processed, so that by swapping the current job, more time than necessary for the completion of both jobs

would elapse. Schrage [45] considered a variation of the algorithm in order to cope with problems like these. He also calculated the mean number of preemptions per job as well as the residence time in the system.

However, it is difficult to estimate the processing time, required for a request at the time of arrival, and therefore the SRPT and SPT disciplines do not seem to be suitable in time-sharing systems.

3.2.5 Two Algorithms by Coffman for Limited Swapping.

Coffman [12] proposes two approaches for modifying the RR and multilevel scheduling algorithms so as to limit or reduce swapping. Firstly, the service discipline must be altered so that programs, which have already received a certain amount of processing, would no longer be swapped when they are processed again. Secondly, the quantum allowance may depend, either on the state of the system, or on its rate of change. Thus, when the number of requests increases, the quantum also increases until service is completed, or there are no new arrivals. In times of high input rates there is a minimum of swapping. When normal input is experienced, little or no difference is seen as a result of this modification.

Let us consider the multiple level feedback model discussed by Coffman and Kleinrock, (see Fig. 8). This model is referred to as the FB_N when there are N levels of queues for which interarrival and service times are exponential. Hence $A(t)$, the interarrival time distribution is given by

$$A(t) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0, \quad \lambda > 0 \\ 0 & t < 0, \quad \lambda > 0 \end{cases}$$

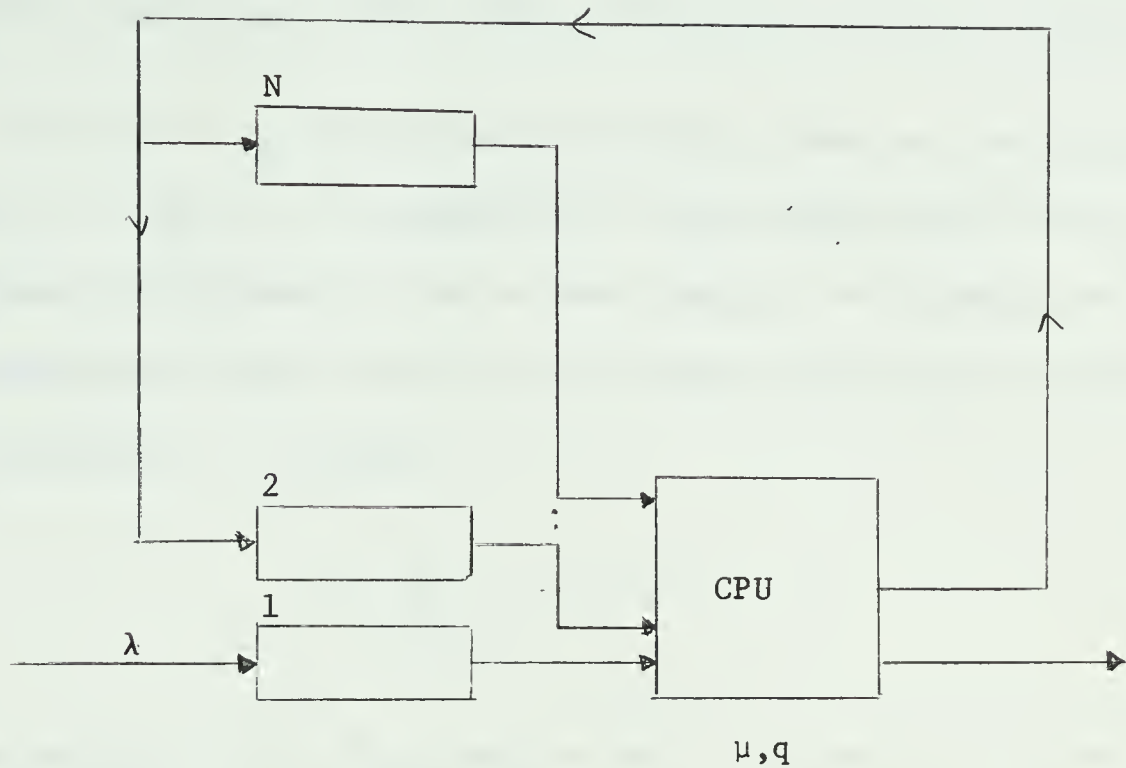


Figure 8. The FB_N model

The service time τ has distribution $B(\tau)$ where

$$B(\tau) = \begin{cases} 1 - e^{-\mu\tau} & \tau \geq 0, \quad \mu > 0 \\ 0 & \tau < 0, \quad \mu > 0 \end{cases}.$$

Here the unit at the service point of any queue level will be serviced if and only if the lower queues are all empty. When a unit has completed its quantum and still needs another, it will be returned to the next higher level queue. The N^{th} level queue, like the others, operates on a FCFS basis but there are no quantum controls (i.e. once a unit gets attention, it is serviced to completion). If the time quantum, q , tends to zero, then for a finite N , we have an FCFS system since the lower $(N-1)$ queues will be serviced in an infinitely small time and the N^{th} level units will then be served, for an infinite number of q 's, that is, to completion (by definition of the FCFS discipline).

If q is relatively large compared with the processing time t , the service will tend to the FCFS (with service to completion) discipline.

With the FB_N model described above, there are two controllable parameters, q and N . An extension of this system would be the introduction of priorities at the various levels. For example, we can assume independent Poisson inputs at each level with average arrival rate λ_p units/sec. We define

$$\lambda = \sum_{p=1}^{\infty} \lambda_p \quad \text{for } \tau < \infty.$$

Further extension to the FB_N would be the allocation of different quanta to different queue levels and different mean service times to different priority levels. Schrage [45] has considered the case of quantum sizes for any level depending on the number of units in it. Whilst these modifications seem quite simple, they introduce grave computational and analytical difficulties. For the simplest FB_N model, i.e. the same quantum for all levels with no priorities on arrival, and $q \rightarrow 0$ the response time is

$$T(t) = \begin{cases} \frac{1/\mu}{1-\rho} & N < \infty \quad (A) \\ \frac{\frac{\lambda}{2} \int_0^t x^2 dF(x)}{[1-\rho(1-e^{-\mu t})]^2} + \frac{t}{1-\rho(1-e^{-\mu t})} & N \text{ at } \infty \quad (B) \end{cases}$$

where

$$F(x) = \begin{cases} 0 & x < 0 \\ 1-e^{-\mu x} & 0 \leq x < t \\ 1 & x \geq t \end{cases}$$

(A) corresponds to the FCFS system and (B) to a preemptive discipline. Here $1/\mu$ sec. is the mean service time, λ the arrival rate, $\rho = \frac{\lambda}{\mu}$ and t is the required service time. If the number of queue levels is limited to two, then $FB_N = FB_2$ and we have the Foreground-Background (FB) model [12]. In this model, any unit requiring more than one quantum is fed back to the Background queue and is served to completion whenever it gets service next. Those units requiring only one quantum will be served in the Foreground queue. For this model we shall consider the time spent waiting in the queues. If W_F and W_B are the mean waiting times in the Foreground and Background queues respectively, Coffman [12] showed that

$$W_F = \frac{\lambda \delta^2 Q}{(1-\lambda)(1-\delta)^2}$$

and

$$W_B = \frac{\lambda \delta Q / (1-\delta)^2}{(1-\lambda)(1 - \frac{\lambda}{1-\delta})} + \frac{\lambda Q}{1-\delta} ,$$

where δ is the probability that an arrival requires more than one quantum of service to complete his request, and λ is the probability that an arrival occurs in quantum Q .

Generalizing the FB model discussed above to include multiple-quantum allocation, let us denote by W_k the mean waiting time in a queue for programs requiring k quanta of service, then we have

$$W_k = \lambda^{k-1} W_F + (1-\lambda^{k-1}) W_B(k)$$

$W_B(k)$ is the mean waiting time for a k -quanta job which has been fed back to the background queue, and

λ^{k-1} is the probability that a k -quanta job will be serviced to completion in the foreground queue.

The average length of time spent in the foreground queue then is $Q/(1-\lambda\delta)$ instead of simply Q .

τ_k is the mean number of quanta spent in the foreground queue by a k -quanta job which is served to completion in the background queue.

$$\tau_k = \frac{1}{1-\lambda} - \frac{k\lambda^k}{1-\lambda^k}$$

The following results were obtained by Coffman [12] for the single quantum (SQ) and multiple quantum (MQ) allocations:

- i) For $q = 0.5$ secs, there is no appreciable difference in the mean waiting time.
- ii) The preferential treatment for short jobs is seen in both.
- iii) For jobs requiring only one quantum the SQ is much better, for 2 and 3 quanta jobs the MQ is more efficient, and for jobs of 4 and more quanta, they are about the same.

CHAPTER IV

CP/67 TIME-SHARING SYSTEM

4.1 The CP/67 Operating System.

The Control Program-67/Cambridge Monitor System (CP-67/CMS), developed at the Cambridge Scientific Center, Massachusetts, is a set of control and processing programs which were written for an IBM/360, Model 67. CP/CMS, as stated in Section 1.1.3, is based on the concept of virtual machines [29].

4.1.1 Design Goals.

CP/CMS, in addition to putting all the IBM/360 facilities at the disposal of the user, is designed so that [17]:

- i) several users can "simultaneously" carry out a wide range of data processing applications on one system; and,
- ii) each user, via his terminal, can initiate, monitor and terminate his particular application by carrying on a command/response type dialogue or conversation with the system.

4.1.2 Components of the System.

CP/CMS consists of two major components: a control program (CP/67) and a monitor (CMS). CP/67 is a time-sharing, multiprogramming, executive program which, in conjunction with CMS, provides a general purpose time-sharing system. CMS creates the conversational capabilities

for users to develop, debug and execute programs from remote terminals. Both CP/67 and CMS can function separately on the appropriate IBM/360 configuration (see 4.1.3). To realise their full potential, they must be run together, in which case CMS functions under the supervision of CP/67. The languages available under the System include OS/360, Assembler (F) , Fortran (G) , and PL/1 (F) .

4.1.3 Hardware Environment.

In the previous section it is stated that CP/67 can operate on an appropriate IBM/360 configuration. Hence, let us list the minimum hardware required.

- 1 1403 Printer,
- 1 2067-1 or 2067-2 CPU ,
- 2 2311 Disk Drives or 2314 Storage Unit,
- 1 2365 Core Storage Unit,
- 1 2540 or 2501 Card Reader,
- 1 2540 Card Punch,
- 1 2702 Transmission Control Unit,
- 1 1052 On-Line Console Typewriter.

The following optional devices are also supported:

- 1051/1052 Data Communications System,
- 2250 Graphic Display Unit,
- 2820/2301 Drum Storage Controller(s)/Unit(s),
- 2400 Series Magnetic Tape Units,
- 2741 (-1,-2) Communications Terminals.

4.1.4 Present Implementation at University of Alberta.

The CP/CMS System, implemented at the University of Alberta, supports up to 60 virtual machines. This number does not represent the maximum capability of the system but rather is the maximum number of data-lines currently available to the system. Through the use of drum storage rather than disk storage, and with the provision of more lines, the system could probably support more virtual machines as efficiently as its present operation. CMS supports the following language processors; 360 Assembler, Fortran (G) , Snobol, Bruin, AED-0 (Algol) and PL/1, whilst COBOL is soon to be implemented. APL, the conversational language developed by Iverson [30], is running concurrently with CMS under CP.

4.2 The CP/67 Supervisor.

The CP/67 utilizes the virtual machine (VM) concept (see Section 2.2) to provide for each user the facilities of an IBM/360 Model 65. This use of VM's in CP/67 will now be discussed in some detail since some knowledge of it makes for a better understanding of the role of the Dispatcher and, hence, of the model developed.

4.2.1 Virtual Machines in CP/67.

A virtual computing system, simulating hardware facilities, enables the user to create the particular hardware and software environment he desires. For example, he might load and use a specific software system, such as Operating System/360. In this way the user, not the system, determines the facilities available to him.

A potential user of CP/CMS must have some means of identification, i.e. a USERID , with which he "logs in" or "signs on". He then has at his disposal a virtual machine with its virtual address space. Through this he gains access to the standard set of facilities of a real machine. In fact, CP/67 creates for each user at his remote terminal a simulated Model 65 computing system. This the CP/67 accomplishes by:

- i) scheduling and allocating main storage space, CPU time and I/O devices to the virtual computers;
- ii) handling all interrupts;
- iii) protecting system files, users' programs, and users' data during execution;
- iv) keeping statistics on the use and performance of the "real" system.

A user's virtual machine can be in one of three modes:

- i) active (or running), in which case some section of his program is currently being processed by the real CPU;
- ii) passive (or non-runnable) where all of its programs including data are in main and auxiliary memory and the servicing of some interrupt is awaited; and
- iii) runnable, when the virtual machine is in, or waiting to enter, a queue to receive some portion of its CPU time-slice.

During the procedure of "logging a user on" the CP/67 supervisory system must perform the following tasks:

- i) checking USERID and password;
- ii) allocating and initializing the primary user control table - UTABLE ;
- iii) allocating and initializing the segment table (SEGTABLE), page tables (PAGTABLE) and corresponding swap tables (SWPTABLE);
- iv) reserving direct access storage space for page swapping;
- v) creating virtual I/O blocks to describe the user's virtual machine; and,
- vi) chaining virtual device blocks to real device blocks.

The log-off procedure then includes:

- i) closing the reader, printer and punch files;
- ii) disconnecting the user's virtual machine, i.e. setting VMSTATUS to INLOGOFF ;
- iii) releasing data-line and storage for virtual device blocks; and,
- iv) giving the time of the day and also the TIMEUSED in the session.

Before discussing some of the algorithms of CP/67 (e.g. Sections 4.2.4, 4.2.5 and 4.3) let us describe some of the mapping tables mentioned above.

4.2.2 Description of Some Mapping Tables.

For each user, a UTABLE is set up with which all other user

blocks are associated. Along with the virtual I/O blocks, the UTABLE completely indicates the status of the virtual machine. Some of the information included in the UTABLE is:

- i) a pointer to the user's SEGTABLE ;
- ii) VMACHSIZ, the size of the VM ;
- iii) VCHCOUNT, the number of virtual selector channels attached;
- iv) VMSTATUS, the state of the VM e.g. PAGEWAIT, IOWAIT, etc.;
- v) TIMEUSED, time used by VM since log on ;
- vi) NEXTUSER, pointer to next user's UTABLE ;
- vii) $U * 4$ (PRCLASS), user privilege class and priority class.

The importance of the UTABLE in CP/67 can be accurately gauged from the above subset of information contained in it. As (i) indicates a SEGTABLE is created for each user and in turn each entry corresponds to a page table (PAGTABLE). Hence, a SEGTABLE entry contains the number of page table entries and the initial address of the page table; and, a PAGTABLE entry (one for each page) contains the real address of the page in core, if it is in core, and the in-core indicator. Corresponding to each PAGTABLE entry there exists a SWPTABLE entry to facilitate the swapping of pages. This entry contains the physical storage address of pages not resident in core and the USERID of the owner of the page. The last table to be considered is the CORTABLE, which is created at SYSGEN (system generation) and for which each entry refers to a block of 4096

bytes in real core. Its size depends on that of real memory and the relative positions of its entries indicate the core addresses of the pages described. This table contains pointers to the corresponding SWPTABLE entry for the virtual page which currently occupies the real page and also a bit showing whether a page is locked in core or not.

Thus the direct correspondence between CORTABLE and SWPTABLE, SWPTABLE and PAGTABLE, and SEGTABLE and PAGTABLE indicates the intricate addressing necessary (see Figure 9).

4.2.3 Virtual Addressing.

Any direct reference to the storage addresses of data structures by a programmer in his VM will actually be to the virtual and not the physical addresses. The correspondence between these addresses, virtual and physical, is maintained by the control program in the address mapping tables. Some writers, have proposed four-component addressing under paging and segmentation, e.g., Wegner [55] and Arden et al. [2].

In CP/67, three component addressing (i,j,k) is used, where i is the segment address in SEGTABLE, j is the page address in the PAGTABLE, and k is the location of the page in core storage. Since the address mapping tables rather than the virtual addresses are modified during swapping operations, the above addressing scheme is referred to as Virtual Addressing (see Figure 9).

Let us now consider the actual Paging and Swapping algorithms in CP/67.

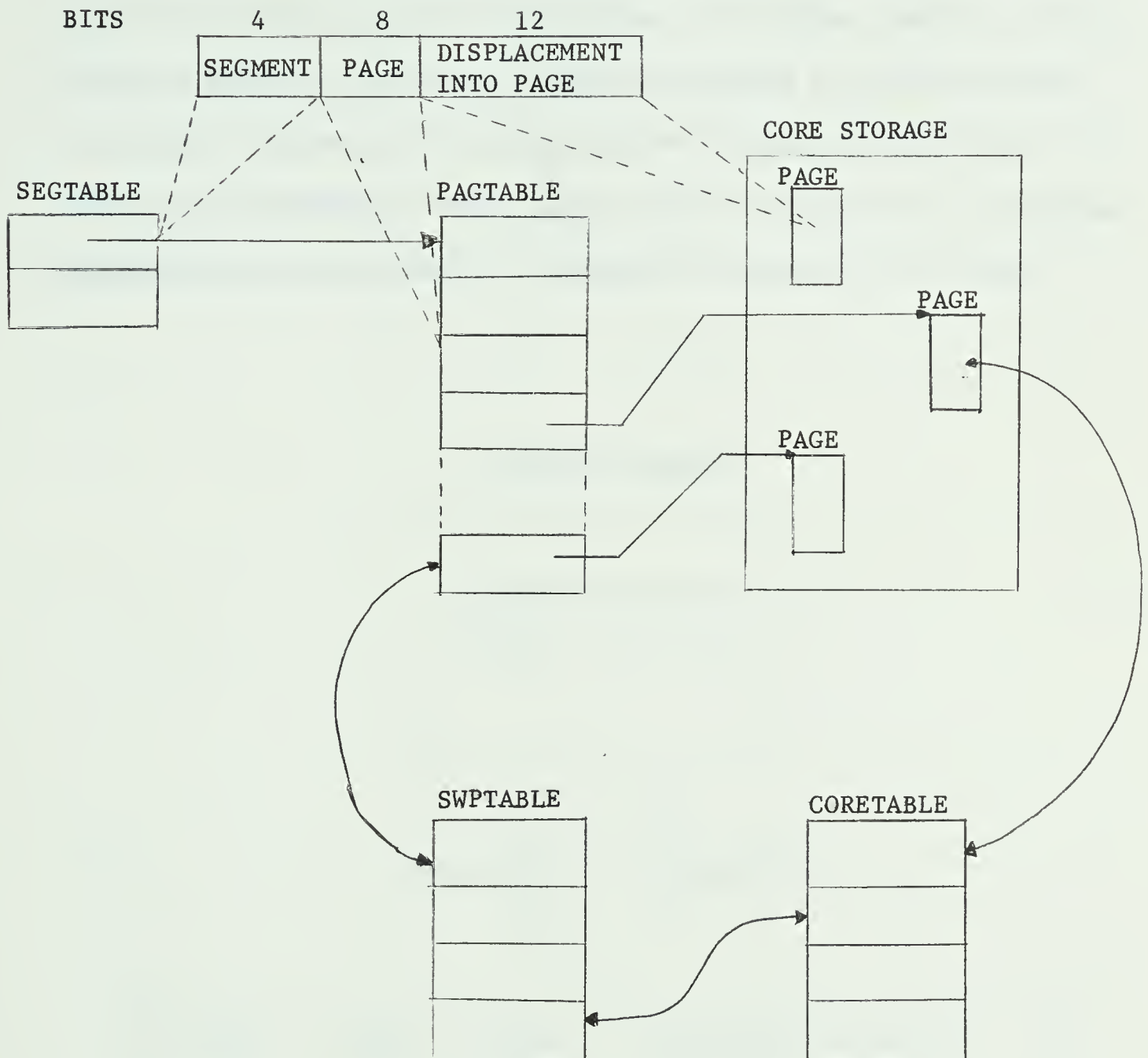


Figure 9. Virtual addressing and address mapping tables.

4.2.4 Paging in CP/67.

When a user logs on, the system places the first page of his program into main storage. [Note: The page size presently used in the U of A's CP/67 system is 4096 bytes.] The other pages are loaded into main memory as required. While his program is being executed, the system is dynamically transferring the virtual addresses into real main storage addresses. These pages need not be stored in contiguous locations and are, in general, scattered throughout main storage.

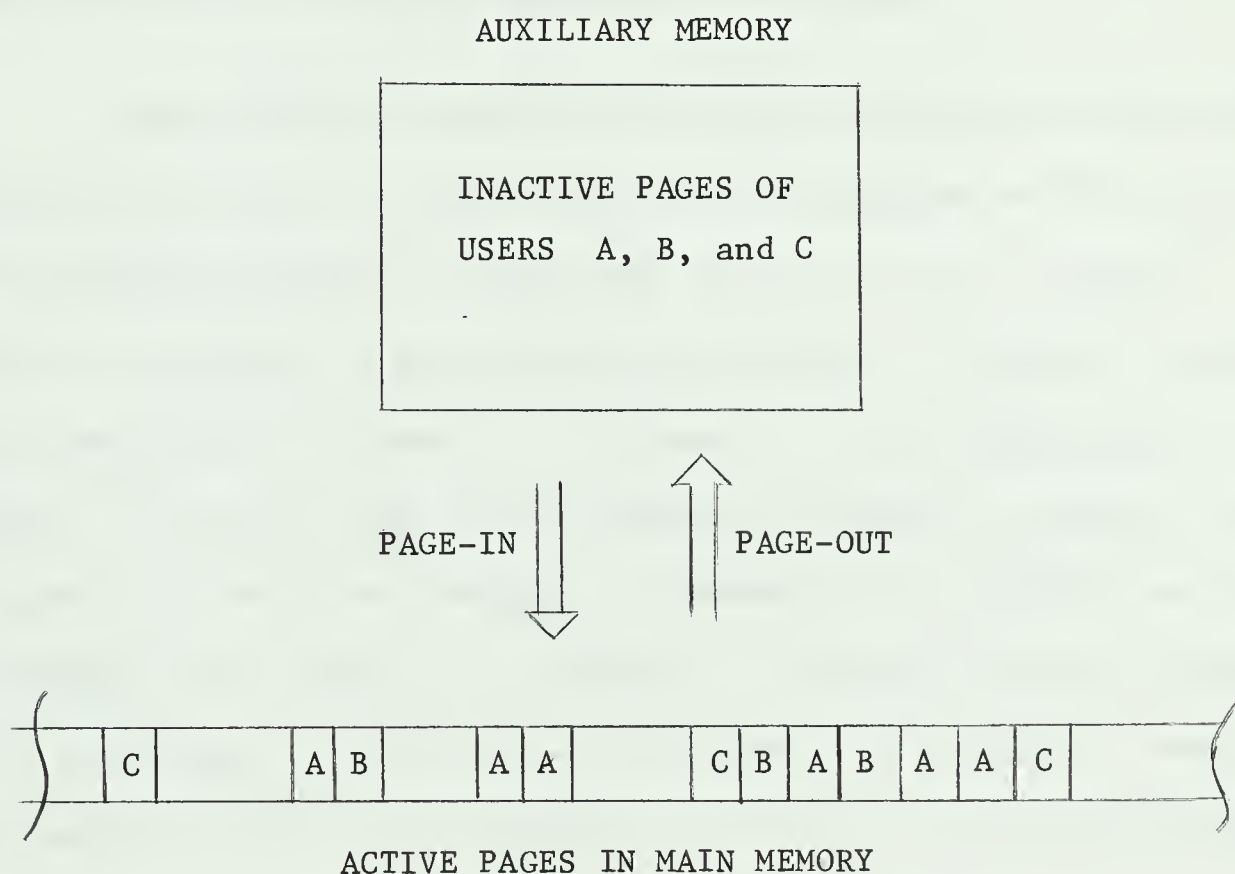


Figure 10. Allocation and movement of pages in memory.

Whenever there is a paging request, an interrupt occurs and a check is made on the status of the particular page, i.e. whether it is core resident, non-core resident or in transit. The system module, PAGTRANS, handles all the paging demands from users' programs. Its functions include "freeing up" main memory space when required, performing necessary I/O operations (i.e. moving pages from main memory to secondary memory and vice versa) and protecting the system against paging overload during peak page demands. To effect the last function, there is a maximum number of paging operations permitted at any time, PAGMUT, and if this is not greater than PAGARP, the number of current paging operations, no paging operations are allowed.

When PAGTRANS translates the virtual address and finds that the page is in core, its other functions are bypassed and the processing of the program continues. If the page is not in core, the user's SWPTABLE is searched to get the physical address of the page in storage and to ascertain if the page is "in transit". If the page is not in transit, the transit flag in the SWPTABLE is changed to indicate that it is now in transit and control is returned to the DISPATCH routine. If the page is in transit, the VMSTATUS is changed to indicate PAGEWAIT and it is brought in as soon as an I/O channel is available. When it is brought into core it is available to the user as soon as his CPRQUEST queue is served.

4.2.5 Swapping Algorithm.

Whereas the paging-in operation is relatively simple, the selection of the page to be removed from core is more controversial.

This was illustrated in an earlier section (see 2.4.2) when consideration was given to a number of proposed swapping algorithms.

In CP/67 each entry in CORTABLE is examined in a round-robin (RR) manner to check the availability of its associated page for swapping. However only those pages which are not locked in core, i.e. LOCK Cnt byte in CORTABLE is zero, are eligible. The selection of a page is a two-pass process. Each entry is examined in the first pass and the corresponding page is selected if either;

- i) the UTABLE pointer indicates the page is unused; or
- ii) the USERID associated with the page is neither in Q_1 nor in Q_2 of the dispatcher (see Section 4.3.2).

If no eligible page has been found, the second pass is entered in which any unlocked page is selected.

The swapping algorithm actually uses the RANDOM selection on a two-priority basis. That is, unused pages or those belonging to non-runnable users are selected first and afterwards, if necessary, any unlocked page is selected. It would be useful to compare this algorithm with the FIFO selection but this is outside the scope of this research.

4.3 The Dispatcher in CP/67.

The control routine, DISPATCH in CP/67, receives control after the interrupt handling routines have completed their processing. The time-used when the system is in the supervisor state is charged to the user causing the interrupt or, in the case of a systems interrupt, to

the current user and it is added to the number in the TIMEUSED entry in his UTABLE. Prior to the running of any user's program (i.e. CPU service given), all pending interrupts and requests are handled.

Requests for service are made by the users' programs, that is, in their VM's. When they receive the attention of the CPU, they are processed for the desired service time in a series of time quanta. In the CP/67 system implemented at U of A, the quantum size is set at 50 m-seconds. There are two queues of users' requests denoted by Q_1 , the interactive users' requests, and Q_2 , the longer service time, production-type requests. Subsequently requests will be referred to as Q_1 or Q_2 requests and users as Q_1 or Q_2 users according to whether the service time required is of the shorter or longer class. These terms " Q_1 user" and " Q_1 request", etc., are synonymous.

The service of any requests, whilst in Q_1 or Q_2 , is subject to interrupts such as I/O, paging, etc. This division of requests into Q_1 and Q_2 according to the amount of processing required is an attempt to reduce the overhead due to paging. In this manner only a subset of current users will be able to gain control of the system at any one time. Furthermore there may be maximum limits imposed on the number of requests permitted to be in Q_1 and Q_2 at any time.

4.3.1 Users' States.

A user may be in one of two modes, runnable and non-runnable. The non-runnable users are those in:

- i) page-wait,
- ii) I/O operation, or
- iii) CP console function.

whereas, the runnable users must be in one of the following states:

- i) in Q_1 - getting or awaiting service,
- ii) waiting to get into Q_1 ,
- iii) in Q_2 ,
- iv) waiting to get into Q_2 , or
- v) idle, not requiring system resources.

It is important to note that:

- a) state (i) can only be entered from state (ii),
- b) state (iii) can only be entered from state (iv)
- c) states (ii) and (iv) are entered only when there is a
READ operation pending on the terminal or console.

Generally, interactive jobs requiring short processing time with much terminal interaction to complete their requests, are provided with better or faster service in a time-sharing environment than that in a batch processing one. To achieve this, it is necessary for the allocation or sharing of the CPU to be scheduled in a priority-oriented manner. The following order is the order of preference in selecting a user to be processed:

- i) the current (interrupted) user if he is runnable and has not used up his full quantum of 50 m-seconds;
- ii) a runnable user in Q_1 ;
- iii) a runnable user wishing to enter Q_1 ;
- iv) a runnable user in Q_2 ; and
- v) a runnable user wishing to enter Q_2 .

Since non-runnable users cannot be processed, the CPU enters the WAIT state if there are no eligible, runnable users.

4.3.2 Characteristics of Q_1 and Q_2 Users.

In the priority scheduling algorithm distinction is made between Q_1 and Q_2 users. The behaviour and characteristics of these two types of users may be outlined as follows:

Q_1 Users. This queue, Q_1 , is occupied mainly by interactive tasks. However, all tasks must spend some time in Q_1 ; the interactive tasks spend most of their service time here whereas the users with longer service time are feedback to Q_2 for further processing. Thus we have the following situations:

- i) a Q_1 user requiring paging or some I/O operation before the expiration of its 50 m-seconds time slice is feedback to the end of Q_1 or to await entrance to Q_1 again;
- ii) a Q_1 user whose accumulated service time on one request is $\geq .4$ seconds is 'feedback' to enter (or await entrance to) Q_2 ;

- iii) a Q_1 user who completes 50 m-seconds of continuous service without interruption is deemed to be a production-type or compute-bound user and is sent to enter, or await entrance to Q_2 ;
- iv) within Q_1 , the order of service is FCFS.

Q_2 Users. These are compute-bound jobs which require much CPU time but little terminal interaction to complete their processing. With these users the following service-algorithm is used:

- i) a Q_2 user is serviced if there are no eligible Q_1 users and, of course, no outstanding interrupts;
- ii) a Q_2 user, when he gets the attention of the CPU, is serviced in a similar manner to a Q_1 user, i.e. for a maximum of 60 m-seconds time-slice;
- iii) a Q_2 user has his priority number raised by one whenever he receives 5 seconds of accumulated processing time for one CPU request (this is a simplified view of the actual priority assignment in the dispatch); and
- iv) within Q_2 , the user with the lowest priority number receives service first.

In Figure 11, a broad view of the scheduling algorithm of CP/67 is presented. Whereas, in Section 4.3.1, there are distinct states for users waiting to enter Q_1 and Q_2 , in this flow chart and in the subsequent analysis users both in, and waiting to enter Q_1 are treated together. Q_2 users are handled similarly.

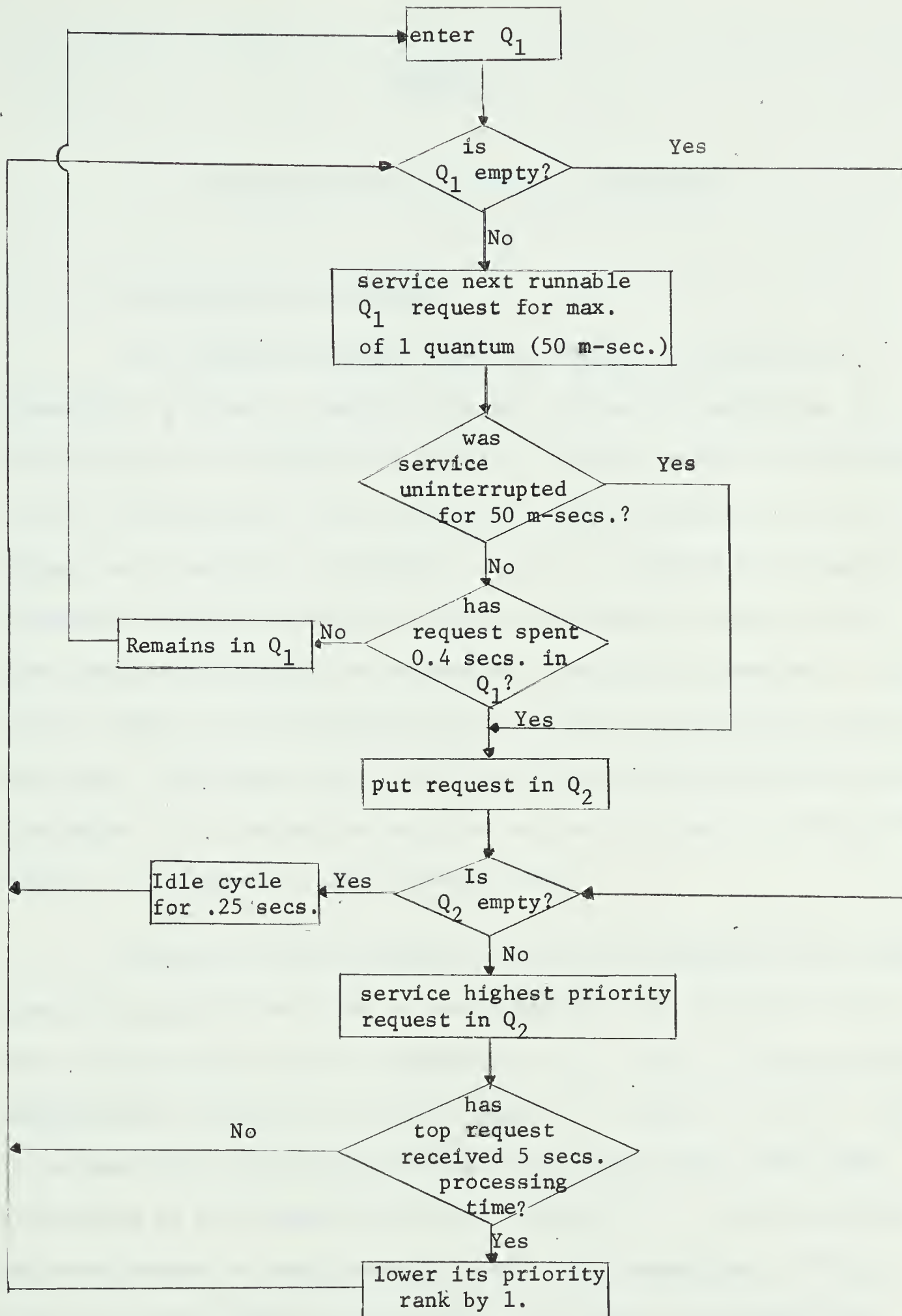


Figure 11. Scheduling logic of dispatcher in CP/67.

CHAPTER V

A QUEUEING MODEL OF THE CP/67 DISPATCHER

5.1 Formulation of the Model.

The following queueing model is proposed to describe the scheduling algorithm in the CP/67 system. The model consists of N priority levels of requests for service. Requests enter the system from outside at two levels: (i) at level 1, the interrupts e.g. I/O, paging, etc. and (ii) at levels 2 to N , requests from users' programs. The input process is taken to be Poisson at each priority level and these processes are assumed to be mutually independent. The service times for the different priority levels have arbitrary distribution functions. The single server, the CPU, serves under a preemptive-resume discipline. If a request has received service in excess of some specified amount, it is feedback a lower priority level.

Within the limits imposed by the above assumptions, the model gives an accurate description of the requests in the first two levels, the interrupts and users' CPU requests in Q_1 . That is, the parameters discussed are accurate for priority levels k , where $k = 1, 2$. There is no input from outside the system for the other levels, that input is provided by the requests which were formerly in Q_1 but which have now been feedback to lower priority levels. The assumption of Poisson input for these levels is only correct in the case of an exponential service time distribution. In spite of this restriction it was considered better to give a general description of the N priority level model

and then obtain expressions for the parameters for the different levels rather than discuss the different levels separately from the beginning.

It may be remarked here that in the study of priority queues the hierarchy of requests can be considered as two levels with respect to, say, the k^{th} priority class ($k > 1$). The Priorities are those requests from priority levels 1 to $k-1$, and the customers are those from the k^{th} level. Requests from the lower priority levels i.e. $k+1, k+2, \dots$ are nonexistent as far as the k^{th} level is concerned.

Moments of the following random variables have been either obtained explicitly or characterized as the solution of some functional equations: the completion time, the busy period, the queue size and the waiting time for any priority level. Some indication is also given of the effect of the quantum size on the number of preemptions for a specified service time distribution as well as the effect of higher priority requests on lower ones.

Let us now define these random variables. In Figures 12 and 13, the interaction between some of them for the k^{th} priority level is illustrated.

Quantum Size. The random variable, Y_k , is the quantum size or length of time slice a k^{th} priority request receives at each processing by the CPU. Let its distribution function be denoted by $Q_k(y)$, where

$$Q_k(y) = P(Y_k \leq y) \quad \text{for} \quad 0 \leq y \leq .05 \text{ secs.}$$

It may be recalled that if a request from priority level k ($k > 1$) requires more than .05 secs. of CPU time, it is recycled to the end of

its respective queue and receives another time slice Y_k , etc., until it has obtained the total service time it needed.

Service Time. The service time, X_k , is the total processing time a k^{th} priority request receives from the CPU. Since in this time-sharing system requests do not seize the CPU for a continuous interval x , X_k will be the sum of service intervals S_{ik} (see Figure 12), which are separated by intervals during which higher priority ($< k$) requests are serviced. The CPU time used by the CP supervisor to service the timer interrupts is negligible and hence the S_{ik} in Figure 12 may well be greater than .05 secs. (i.e. consists of more than one time quantum). The distribution function of X_k is $H_k(x)$, where

$$H_k(x) = P(X_k \leq x), \quad x \geq 0.$$

These service times are assumed to be mutually independent random variables.

Waiting Time. The waiting time, Z_k , is the time a request waits in a queue to receive its first slice of service. This does not include time spent in a queue after it has been fed back to receive more processing.

The distribution function associated with this random variable is $W_k(z)$ where

$$W_k(z) = P(Z_k \leq z), \quad z \geq 0.$$

Completion Time. The completion time, T_k , is the interval from the receipt of the first time interval of CPU processing in the k^{th} priority queue until the task is serviced to completion in that queue and leaves the system or is feedback to priority level $k+1$. It includes the time spent awaiting further processing when it has been feedback into the same k^{th} priority queue. Its distribution function is $C_k(t)$ where

$$C_k(t) = P(T_k \leq t), \quad t \geq 0.$$

Busy Period. The busy period, B_k , is the time interval the computer is continuously busy with tasks of priority $\leq k$. It may be started either with a k^{th} priority task or with a higher priority ($\leq k-1$) task.

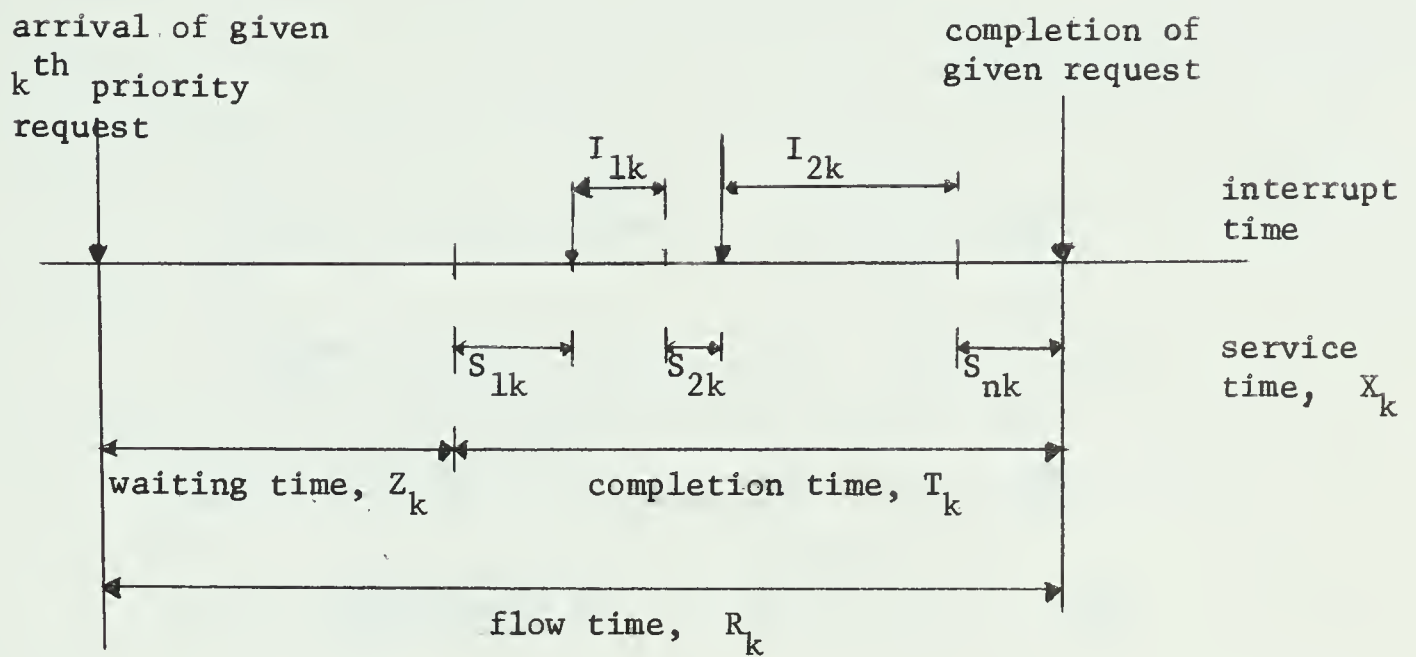
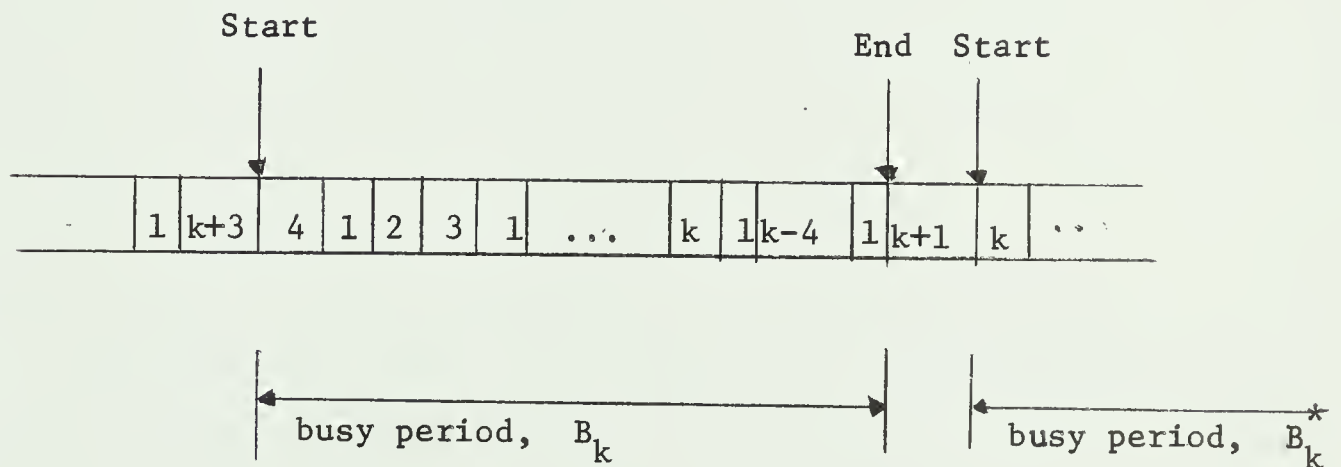
The distribution of the busy period is denoted by $D_k(b)$

where

$$D_k(b) = P(B_k \leq b), \quad b \geq 0.$$

The busy period, B_k^* , initiated by a k^{th} priority request has a distribution function $D_k^*(b)$, where

$$D_k^*(b) = P(B_k^* \leq b).$$

Figure 12. Timing diagram of priority k level.Figure 13. Busy period for the k^{th} priority.

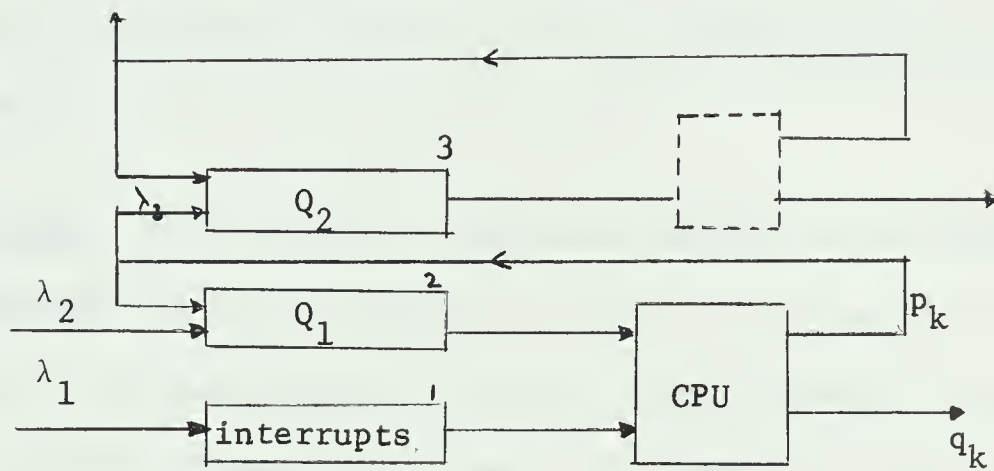


Figure 14. Queueing model of dispatcher.

5.2. Analysis of Random Variables in the Model.

The method of analysis used in this thesis is to consider the queueing process as a stationary one, that is, the probabilities associated with the various random variables are independent of time. In this way, stationary distribution functions are obtained for all the random variables considered.

Input Process. Since any user can issue one CPU request after another in his program via the terminal, the source of requests is assumed to be infinite. In addition the fact that these requests may occur at random intervals points to the use of the Poisson arrival process. This assumption is essential in most applications of queueing theory (see Appendix A).

Let λ_k be the input requests with priority k , where $k = 1, 2, \dots, N$. Then, from Appendix A, the total input rate to the system is

$$\Lambda = \sum_{k=1}^N \lambda_k .$$

For ease of notation Λ_k will refer to the total input rate of request with priorities k or less, then we have

$$\Lambda_k = \lambda_1 + \lambda_2 + \dots + \lambda_k .$$

Note: It may be remarked here that the assumption of Poisson input for $k \geq 3$ only holds in the case of exponential service time and otherwise can only be viewed as an approximation.

Service Process. Requests are serviced in a preemptive-resume manner with service being given in a series of time quanta. These quanta may be constant, or variable with distribution function $Q_k(x)$ for the k^{th} priority level. At the completion of a quantum of service, the request either is completely processed with probability q_k or returns to the queue with probability p_k . At each interaction of a k^{th} priority request with the CPU, the values of p_k , q_k are independent of those assigned after any previous interaction. It is clear that $p_k + q_k = 1$.

There is feedback of requests from one priority level to the next lower one when,

- (i) the service time exceeds the limit permitted in the particular priority level, e.g. when (see Section 4.3)

$$X_k = \begin{cases} 0.4 \text{ secs.} & k = 2 \\ 5.0 \text{ secs.} & k \geq 3 \end{cases}.$$

There is no feedback for the case of $k = 1$ (i.e. the interrupts).

- (ii) the amount of processing in a single time slice is equal to 0.05 secs., i.e. $y_k = .05 \text{ secs.}$

The assumption of preemptive-resume holds, in the physical situation only when the request is an interrupt. For users' requests, the interruption of service by one on another is non-preemptive or head-of-the-line. However, for the random variables the parameters considered in this thesis, the assumption of preemptive-resume is valid as far as the distributions are concerned. (See Appendix C.)

Let $\psi_k(s)$ be the Laplace transform of $H_k(x)$, then

$$\psi_k(s) = \int_0^{\infty} e^{-sx} dH_k(x) \quad (1)$$

and the r^{th} moment of X_k

$$a_k^{(r)} = \int_0^{\infty} x^r dH_k(x) \quad (2)$$

with $a_k = a_k^{(1)}$.

Now the probability that a request will receive g quanta of service is $q_k p_k^{g-1}$. Therefore the number of quanta per service request is geometrically distributed.

From the service process outlined, the service time distribution can be expressed in terms of the quantum size distribution as follows:

$$H_k(x) = q_k \sum_{m=1}^{\infty} p_k^{m-1} Q_k^{*m}(y) \quad (3)$$

where $Q_k^{*m}(y)$ is the m -fold convolution of $Q_k(y)$ with itself and m is the number of quanta the request has been given (i.e. the m r.v.'s Y_k are independent and have the same distribution for $Q_k(y)$).

$$\sum_{k=1}^m Y_k = Q_k^{*m}.$$

Taking the Laplace transform of (3), we have

$$\psi_k(s) = q_k \sum_{m=1}^{\infty} p_k^{m-1} [\theta_k(s)]^m, \quad (4)$$

where $\theta_k(s)$ is the L.T. of $Q_k(y)$.

Then

$$\psi_k(s) = \frac{q_k \theta_k(s)}{1 - p_k \theta_k(s)} \quad (5)$$

and differentiating (5) w.r.t. s , we get at $s = 0$

$$E(X_k) = \frac{E(Y_k)}{q_k} \quad (6)$$

similarly,

$$E(X_k^2) = \{q_k E(Y_k^2) + 2p_k [E(Y_k)]^2\} / q_k^2. \quad (7)$$

In the remainder of this analysis we will deal only with the service time distribution $H_k(x)$ since $Q_k(y)$, the quantum size distribution, can be expressed in terms of $H_k(x)$.

Completion Times for Priority k . Let $\eta_n(k)$ be the virtual waiting time of a k^{th} priority request, i.e. the time the n^{th} arriving request would have to wait if it were from priority k . Then as mentioned earlier for the r.v. Z_k , the stationary distribution of $\{\eta_n(k)\}$ is denoted by $W_k(z)$. Let its corresponding L.T. be $\Omega_k(s)$,

$$\Omega_k(s) = \int_0^\infty e^{-sz} dW_k(z). \quad (8)$$

and the r^{th} moment of Z_k

$$w_k^{(r)} = \int_0^\infty z^r dW_k(z). \quad (9)$$

Because of the preemptive nature of the model, the tasks of

priority $> k$ have no effect on the sequence $\{\eta_n(k)\}$. Thus for tasks of priority k , the completion time is the sum of the total service time of the k^{th} priority tasks and the additional durations resulting from the service of higher priority tasks. To derive the expression of the L.T. of the distribution of the completion time, it will therefore, be necessary to state the L.T. of the busy period distribution function. The latter will be discussed more fully later in this section.

The L.T. of $C_k(t)$, the completion time distribution function is denoted by $\phi_k(s)$, where

$$\phi_k(s) = \int_0^{\infty} e^{-st} dC_k(t) \quad (10)$$

and the r^{th} moment of T_k is

$$c_k^{(r)} = \int_0^{\infty} t^r dC_k(t) \quad (11)$$

Let $D_k(b)$, the busy period distribution function for tasks of priorities $\leq k$, have a L.T. $\Gamma_k(s)$, then

$$\Gamma_k(s) = \int_0^{\infty} e^{-sb} dD_k(b) \quad (12)$$

The r^{th} moment of B_k is given by

$$\delta_k^{(r)} = \int_0^{\infty} b^r dD_k(b) \quad (13)$$

Let I_{ik} be the random variable associated with the length of the i^{th} interruption during the processing of a k^{th} priority request (see Figure 12). Then we have,

$$T_k = X_k + \sum_{i=0}^M I_{ik} \quad (14)$$

where the r.v. M is the number of interruptions.

The L.T. of the completion time is obtained by using Gaver's [28] method. Under conditions of X_k service time and M interruptions we have,

$$E\{e^{-sT_k} | X_k, M\} = e^{-sX_k} E\{e^{-s \sum_{i=0}^{M-1} I_{ik}}\} \quad (15)$$

It can be seen that I_{ik} is also the i^{th} busy period of priority level $\leq k-1$ which provides the interruption to the k^{th} level. Assuming that these busy periods are independent we have

$$E\{e^{-sT_k} | X_k, M\} = e^{-sX_k} [\Gamma_{k-1}(s)]^M \quad (16)$$

The probability of m arrivals during the service time X_k is

$$P\{M = m | X_k\} = e^{-\Lambda_{k-1} X_k} \frac{(\Lambda_{k-1} X_k)^m}{m!} \quad (17)$$

and hence removing the condition of M interruptions we have

$$\begin{aligned} E\{e^{-sT_k} | X_k\} &= e^{-sX_k} \sum_{m=0}^{\infty} \frac{(\Lambda_{k-1} X_k)^m}{m!} [\Gamma_{k-1}(s)]^m e^{-\Lambda_{k-1} X_k} \\ &= \exp \{-sX_k - \Lambda_{k-1} X_k [1 - \Gamma_{k-1}(s)]\} \quad (18) \end{aligned}$$

Removing the condition of X_k , we have

$$E\{e^{-sT_k}\} = \int_0^{\infty} \exp\{-sX_k - \Lambda_{k-1}X_k[1-\Gamma_{k-1}(s)]\} dH_k(x) \quad (19)$$

$$\text{i.e.} \quad \phi_k(s) = \psi_k\{s + \Lambda_{k-1}[1 - \Gamma_{k-1}(s)]\} \quad (20)$$

Then the first two moments of the completion time are

$$c_k = c_k^{(1)} = a_k(1 + \Lambda_{k-1}\delta_{k-1}) \quad (21)$$

$$c_k^{(2)} = a_k^{(2)}(1 + \Lambda_{k-1}\delta_{k-1})^2 + \Lambda_{k-1}\delta_{k-1}^{(2)} a_k \quad (22)$$

Busy Periods for Priority k . The busy period of priority $\leq k$ may be initiated by the arrival of a task with priority $\leq k$ when the CPU is idle. Under the assumption of Poisson input processes at all levels, the arrival task will have priority $\leq k-1$ with probability $\frac{\Lambda_{k-1}}{\Lambda_k}$ or the priority will be the k^{th} with probability $\frac{\lambda_k}{\Lambda_k}$. If the later occurs, a busy period B_k^* will be generated where $\gamma_k(s)$, the L.T. of the distribution function $D_k^*(b)$ is given by

$$\gamma_k(s) = \phi_k\{s + \lambda_k(1 - \gamma_k(s))\} \quad (23)$$

This result has been obtained by Gaver [28], where B_k^* refers to his so-called Active Interruption process.

When the busy period is started by a task of priority $\leq k-1$, the initial busy period B_{k-1} will have distribution function $D_{k-1}(b)$ and this will be extended by busy periods B_k^* for all the k^{th} priority

arrivals while the server is still busy i.e. with priority $\leq k-1$ tasks. The busy period then is $B_{k-1} + B_k^*$, where B_{k-1} and B_k^* are independent r.v.'s. Using Gaver's [28] approach we have

$$E\{e^{-s(B_{k-1}+B_k^*)} \mid B_{k-1}\} = e^{-sB_{k-1}} \exp\{-\lambda_k B_{k-1} + \lambda_k B_{k-1} \gamma_k(s)\} . \quad (24)$$

Now B_{k-1} has a distribution function $D_{k-1}(b)$, and removing the condition of a busy period B_{k-1} , we have

$$\begin{aligned} E\{e^{-s(B_{k-1}+B_k^*)}\} &= \int_0^\infty \exp\{-sB_{k-1} - \lambda_k B_{k-1} + \lambda_k B_{k-1} \gamma_k(s)\} dD_{k-1}(b) \\ &= \Gamma_{k-1}\{s + \lambda_k(1 - \gamma_k(s))\} . \end{aligned} \quad (25)$$

Now the L.T. of the distribution function $D_k(b)$ can be written

$$\Gamma_k(s) = \frac{\lambda_{k-1}}{\lambda_k} \Gamma_{k-1}\{s + \lambda_k(1 - \gamma_k(s))\} + \frac{\lambda_k}{\lambda_k} \gamma_k(s) . \quad (26)$$

Let $d_k^{(r)}$ be the r^{th} moment of $D_k^*(b)$, i.e.

$$d_k^{(r)} = \int_0^\infty b^r dD_k^*(b) , \quad (27)$$

and $\delta_k^{(r)}$ be the r^{th} moment of $D_k(b)$, i.e.

$$\delta_k^{(r)} = \int_0^\infty b^r dD_k(b) . \quad (28)$$

Then the first two moments of $D_k^*(b)$ can be obtained as follows:

$$\begin{aligned}
 d_k &= d_k^{(1)} = (-1) \frac{d}{ds} \gamma_k(s) \Big|_{s=0} \\
 &= \frac{c_k}{1 - \lambda_k c_k}
 \end{aligned} \tag{29}$$

$$\begin{aligned}
 d_k^{(2)} &= \frac{d^2}{ds^2} \gamma_k(s) \Big|_{s=0} \\
 &= \frac{c_k^{(2)}}{(1 - \lambda_k c_k)^3}
 \end{aligned} \tag{30}$$

which agree with Chang's [6] results. Similarly the first two moments of $D_k(b)$ are,

$$\begin{aligned}
 \delta_k &= \delta_k^{(1)} = (-1) \frac{d}{ds} \Gamma_k(s) \Big|_{s=0} \\
 &= \frac{\lambda_{k-1}}{\lambda_k} \delta_{k-1} (1 + \lambda_{k-1} d_{k-1}) + \frac{\lambda_k}{\lambda_k} d_k
 \end{aligned} \tag{31}$$

$$\begin{aligned}
 \delta_k^{(2)} &= \frac{d^2}{ds^2} \Gamma_k(s) \Big|_{s=0} \\
 &= \frac{\lambda_{k-1}}{\lambda_k} [\delta_{k-1} \lambda_k d_k^{(2)} + \delta_{k-1}^{(2)} (1 + \lambda_{k-1} d_{k-1})^2] + \frac{\lambda_k}{\lambda_k} d_k^{(2)}.
 \end{aligned} \tag{32}$$

Stationary Waiting Time Distribution for Priority k . In this stationary process, the n^{th} arriving task with priority k has a waiting time, Z_k , before its service can begin. This waiting time, Z_k , consists of two random variables; Z_k^* , the time to process all tasks with priority $\leq k$ already in the system, and Z_k^{**} , the time to process all higher priority ($\leq k-1$) tasks arriving during Z_k^* .

The distribution function, $W_k^*(z)$, of the random variable Z_k^* has a L.T. $\Omega_k^*(s)$ which, as Miller [35] showed, can be expressed as

$$\Omega_k^*(s) = \frac{1 - \Lambda_k E(X_k)}{1 - \Lambda_k \left(\frac{1 - \psi_k(s)}{s} \right)}, \quad (33)$$

where $\psi_k(s)$ is the L.T. of the service time distribution function $H_k(x)$.

Let us now consider the alternative approach to the description of the first $k-1$ priority levels as suggested by Welch [56]. The input to these levels will be taken as Λ_{k-1} where the priority of a task is determined by probabilities $\frac{\lambda_1}{\Lambda_{k-1}}, \dots, \frac{\lambda_{k-1}}{\Lambda_{k-1}}$. Similarly the sequence of service times can be considered in the order of arrival. Let $F_{k-1}(x)$ be the distribution function of the service time of the n^{th} arriving task with priority $\leq k-1$. Then we have

$$F_{k-1}(x) = \sum_{i=1}^{k-1} \frac{\lambda_i}{\Lambda_{k-1}} H_i(x). \quad (34)$$

Now, as Welch [56] shows, Z_k^{**} is equivalent to the initial busy period of a single server queueing system with input density Λ_{k-1} and service time distribution $F_{k-1}(x)$. The L.T. $\Omega_k(s)$, of the waiting time distribution function is then found as

$$\Omega_k(s) = \Omega_k^* \{s + \Lambda_{k-1}(1 - \Gamma_{k-1}(s))\}. \quad (35)$$

The r^{th} moment of this distribution function cannot be obtained by differentiation. However, Miller [35], obtained the first two moments

which can be verified by those obtained by Conway et al. [18] with a recursive method. They are

$$E(Z_k) = \frac{\sum_{i=1}^k \lambda_i E(X_i^2)}{2(1 - \sum_{i=1}^{k+1} \rho_i)(1 - \sum_{i=1}^k \rho_i)}, \quad (36)$$

and

$$\begin{aligned} E(Z_k^2) = & \frac{\sum_{i=1}^k \lambda_i E(X_i^3)}{3(1 - \sum_{i=1}^{k-1} \rho_i)^2(1 - \sum_{i=1}^k \rho_i)} + \frac{\left[\sum_{i=1}^k \lambda_i E(X_i^2) \right]^2}{2(1 - \sum_{i=1}^{k-1} \rho_i)^2(1 - \sum_{i=1}^k \rho_i)^2} \\ & + \frac{\left[\sum_{i=1}^k \lambda_i E(X_i^2) \right] \left[\sum_{i=1}^{k-1} \lambda_i E(X_i^2) \right]}{2(1 - \sum_{i=1}^{k-1} \rho_i)^3(1 - \sum_{i=1}^k \rho_i)}, \end{aligned} \quad (37)$$

where $\rho_i = \lambda_i E(X_i)$.

Stationary Distribution of Queue Size. Let us denote by $\xi_n(k, t)$ the queue size of the k^{th} priority requests at time t , that is, the number of requests waiting or being served at time t . Let $\xi_n(k)$ be the queue size immediately after the n^{th} request of priority k has been serviced. The distribution of $\xi_n(k, t)$ for a stationary process, i.e. when $\xi_n(k, t)$ has the same distribution for all $t \geq 0$, will be obtained in terms of the completion time. (See Chang [6].)

Let

$$P_{k,r}^{(n)} = P[\xi_n(k) = r], \quad (38)$$

the the probability generating function (p.g.f.) of $p_{k,r}^{(n)}$ is $U_k(z)$ where

$$\begin{aligned} U_k(z) &= E\{z^{\xi_n(k)}\} \\ &= \sum_{r=0}^{\infty} p_{k,r}^{(n)} z^r. \end{aligned} \quad (39)$$

For the stationary process, we have

$$0 < U_k(0) < 1, \quad (40)$$

and also that $U_k(z)$ satisfies the following relation

$$\begin{aligned} U_k(z) &= \left\{ \frac{[U_k(z) - U_k(0)]}{z} + \frac{\lambda_k}{\Lambda_k} U_k(0) \right\} \phi_k[\lambda_k(1-z)] \\ &\quad + \frac{\Lambda_{k-1}}{\Lambda_k} U_k(0) \Gamma_{k-1}[\lambda_k(1-z)]. \end{aligned} \quad (41)$$

Since $U_k(1) = 1$, by differentiating (41) w.r.t. z and then setting $z = 1$, we have

$$U_k(0) = \frac{1 - \lambda_k c_k}{1 - \frac{\Lambda_{k-1} \lambda_k}{\Lambda_k} (c_k - \delta_{k-1})}. \quad (42)$$

Let

$$V_n(k) = \begin{cases} 1, & \text{if } n^{\text{th}} \text{ departing request is from the } k^{\text{th}} \text{ priority} \\ 0, & \text{otherwise.} \end{cases}$$

From the stationary process, let

$$G_k(z) = P\{V_n(k) = 1\} E\{z^{\xi_n(k)} | V_n(k) = 1\}, \quad (43)$$

The using the expression for $U_k(z)$ in (41), $G_k(z)$ becomes

$$G_k(z) = \phi_k[\lambda_k(1-z)] \left\{ \frac{[U_k(z) - U_k(0)]}{z} + \frac{\lambda_k}{\Lambda_k} U_k(0) \right\}, \quad (44)$$

and

$$G_k(1) = 1 - \frac{\Lambda_{k-1}}{\Lambda_k} U_k(0). \quad (45)$$

Upon normalizing the p.g.f. of $P_{k,r}^{(n)}$ given $V_k(k)$, we have

$$\begin{aligned} U_k^*(z) &= \frac{G_k(z)}{G_k(1)} \\ &= \frac{\phi_k[\lambda_k(1-z)] \left\{ \frac{[U_k(z) - U_k(0)]}{z} + \frac{\lambda_k}{\Lambda_k} U_k(0) \right\}}{1 - \frac{\Lambda_{k-1}}{\Lambda_k} U_k(0)}. \end{aligned} \quad (46)$$

Further, for $|z| \leq 1$, we have

$$U_k^*(z) = \Omega_k[\lambda_k(1-z)] \phi_k[\lambda_k(1-z)]. \quad (47)$$

The average queue size, L_k , is given by

$$\begin{aligned} L_k &= \left. \frac{d}{dz} U_k^*(z) \right|_{z=1} \\ &= -\lambda_k [\Omega_k'(0) + \phi_k'(0)] \\ &= \lambda_k [E(T_k) + E(Z_k)] \\ &= \lambda_k E(R_k), \end{aligned} \quad (48)$$

where R_k is the average flow time for k^{th} priority request. This result holds for stationary queueing processes under general conditions,

(see Saaty [43]). Using expressions (21) and (36), we get

$$L_k = \lambda_k \left[\frac{E(X_k)}{(1 - \sum_{i=1}^{k-1} \rho_i)} + \frac{\sum_{i=1}^k \lambda_i E(X_i^2)}{2(1 - \sum_{i=1}^{k-1} \rho_i)(1 - \sum_{i=1}^k \rho_i)} \right], \quad (49)$$

where $\rho_i = \lambda_i E(X_i)$.

Effect of Priority Scheduling. The quantum size, Y_k , for a k^{th} priority task may be considered to be extended when higher priority tasks interrupt its service. An example of this is provided by the time to handle timer interrupts which has generally been assumed to be zero. If the random variable, ϵ_k , represents this increase, then the new quantum size which includes these interrupts $\hat{Y}_k$, can be expressed as

$$\hat{Y}_k = Y_k + \epsilon_k. \quad (50)$$

Let the distribution function of $\hat{Y}_k$ be $\hat{Q}_k(y)$ where

$$\hat{Q}_k(y) = P(\hat{Y}_k \leq y). \quad (51)$$

Also let the service time X_k , be extended by η_k where

$$\hat{X}_k = X_k + \eta_k \quad (52)$$

so that $\hat{X}_k$ is the new service time.

Now $\hat{H}_k(x)$, the new service time distribution, can be expressed in terms of $\hat{Q}_k(y)$, similar to (3). Then taking the L.T., we get

$$\hat{\psi}_k(s) = \frac{q_k \hat{\theta}_k(s)}{1 - p_k \hat{\theta}_k(s)} , \quad (53)$$

where $\hat{\psi}_k(s)$, $\hat{\theta}_k(s)$ are the L.T.'s of the distribution function of $\hat{X}_k$ and $\hat{Y}_k$, respectively.

Differentiating (53) w.r.t.s., we get

$$E(\hat{X}_k) = \frac{E(\hat{Y}_k)}{q_k} . \quad (54)$$

Since

$$E(\hat{Y}_k) = E(Y_k) + E(\epsilon_k) , \quad (55)$$

then we have

$$E(\hat{X}_k) = E(X_k) + \frac{E(\epsilon_k)}{q_k} . \quad (56)$$

Thus if the increase in the quantum size can be determined, then the corresponding increase in the expected service time can be easily obtained.

5.3 Analysis for Q_1 , that is, Priorities ≤ 2 .

The model presented in Sections 5.1 and 5.2 does not make any other restrictions on the service time distribution except that its L.T. should exist. However to analyze Q_1 numerically, it is necessary to make some assumptions about the distribution function of the service times and quantum sizes for $k \leq 2$. The unit of measurement is 1 second throughout this analysis. Let us suppose that, for $k \leq 2$, the quantum size is uniformly distributed so that

$$Q_k(y) = \begin{cases} \frac{1}{.035} (y - .015) & .015 \leq y \leq .050 \\ 0 & \text{otherwise} \end{cases} .$$

Let the Poisson input densities of the first two queues be 10 and 4 respectively. Further let $H_k(x)$, the service time distribution, be exponential, i.e.

$$H_k(x) = \begin{cases} 1 - e^{-\mu_k x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

and

$$\mu_1 = 200 \quad \text{and} \quad \mu_2 = 5$$

then

$$\begin{aligned} a_1 &= .005 & a_1^{(2)} &= .000033 \\ a_2 &= .2 & a_2^{(2)} &= .08 \end{aligned}$$

Let ρ_k , where $\rho_k = \lambda_k a_k$, represent the fraction of time the CPU is processing requests from the k^{th} priority queue (i.e. CPU traffic intensity of the k^{th} level). Then the assumption of the above exponential distributions implies that the CPU is servicing interrupts and Q_1 requests for 5% and 80% of the time, respectively. Then from (6), q_k , the probability that a k^{th} priority request is serviced to completion in any quantum is given by

$$q_k = \frac{E(Y_k)}{a_k} \quad \text{for } k \geq 2 .$$

For a Q_1 request, $k = 2$, then

$$q_2 = \frac{E(Y_2)}{a_2}$$

$$= .175 \quad .$$

The average number of times a task ($k = 2$) will be serviced before it is completed is G_k ,

$$G_k = \left[\frac{a_k}{E(Y_k)} \right]$$

where $[y]$ is the smallest integer greater than y .

$$G_2 = 6 \quad .$$

Hence a Q_1 request is serviced 6 times on the average before it is completely processed.

The completion time, T_k , provides a means of estimating what increase in time a k^{th} priority task must remain in the system before it is serviced to completion. The average completion time for a Q_1 request is, from (21) and (29)

$$C_2 = \frac{a_2}{1 - \lambda_1 a_1}$$

$$= .21 \quad .$$

The percentage increase in time to service a Q_1 request to completion is

$$100 \left(\frac{C_2 - a_2}{a_2} \right) = 5\% \quad ,$$

i.e., there is an increase of 5% in the servicing of a Q_1 request.

From (22) and (29), the second moment of T_2 is given by

$$\begin{aligned}
c_2^{(2)} &= \frac{a_2^{(2)}}{(1-\lambda_1 a_1)^2} + \frac{\lambda_1 a_2 a_1^{(2)}}{(1-\lambda_1 a_1)^3} \\
&= .09
\end{aligned}$$

then

$$\text{Var } (T_2) = .05 \quad .$$

For the average busy period of priority 2, δ_2 , we have from (31),

$$\begin{aligned}
\delta_2 &= \frac{\lambda_1 a_1 + \lambda_2 a_2}{(\lambda_1 + \lambda_2)(1 - \lambda_1 a_1 - \lambda_2 a_2)} \\
&= .4 \quad .
\end{aligned}$$

The CPU, therefore, services Q_1 requests and interrupts continuously for an average of .4 seconds. The second moment of B_2 using (32) is

$$\begin{aligned}
\delta_2^{(2)} &= \frac{\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)}}{(\lambda_1 + \lambda_2)(1 - \lambda_1 a_1 - \lambda_2 a_2)^3} \\
&= 6.8 \quad .
\end{aligned}$$

and so

$$\begin{aligned}
\text{Var } (B_2) &= 6.8 - .16 \\
&= 6.6 \quad .
\end{aligned}$$

We may also calculate the average length of the busy periods which were initiated by a Q_1 request. This is given by d_2 , hence by (29),

$$\begin{aligned}
d_2 &= \frac{a_2}{1 - \lambda_1 a_1 - \lambda_2 a_2} \\
&= 1.33 \quad .
\end{aligned}$$

The second moment is $d_2^{(2)}$, from (30) we have,

$$\begin{aligned} d_2^{(2)} &= \frac{c_2^{(2)}}{(1 - \lambda_2 c_2)^3} \\ &= \frac{a_2^{(2)}(1 - \lambda_1 a_1) + \lambda_1 a_2 a_1^{(2)}}{(1 - \lambda_1 a_1 - \lambda_2 a_2)^3} \\ &= 22 . \end{aligned}$$

The average duration of busy period started with a Q_1 request being served is three times that begun by an interrupt.

The average waiting time for a Q_1 request, w_2 , is given by (36)

$$w_2 = \frac{\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)}}{2(1 - \lambda_1 a_1)(1 - \lambda_1 a_1 - \lambda_2 a_2)} .$$

This waiting time is the time spent by a request before it can receive its first time slice from the CPU. It is therefore dependent on the number of interrupts ($k = 1$) to be processed and the number of Q_1 requests already in the system. The expression derived clearly shows the effect of the second moments of the service time on the expected waiting time. On substituting we get

$$w_2 = 1.12 .$$

A Q_1 request then has an expected waiting time of 1.12 secs. The second moment of Z_n is given by $w_2^{(2)}$, from (37)

$$\begin{aligned}
w_2^{(2)} &= \frac{\lambda_1 a_1^{(3)} + \lambda_2 a_2^{(3)}}{3(1-\lambda_1 a_1)^2(1-\lambda_1 a_1 - \lambda_2 a_2)} + \frac{[\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)}]^2}{2(1-\lambda_1 a_1)^2(1-\lambda_1 a_1 - \lambda_2 a_2)^2} \\
&\quad + \frac{\lambda_1 a_1^{(2)} [\lambda_1 a_1^{(2)} + \lambda_2 a_2^{(2)}]}{2(1-\lambda_1 a_1)^3(1-\lambda_1 a_1 - \lambda_2 a_2)} \\
&= 2.5 \quad .
\end{aligned}$$

The variance of Z_k is

$$\text{var}(Z_k) = 1.25 \quad .$$

The average flow time of a Q_1 request i.e. the total time spent in the queueing system is R_2 where

$$\begin{aligned}
R_2 &= c_2 + w_2 \\
&= 1.33 \quad .
\end{aligned}$$

In this stationary process, the number of Q_1 requests in the queue is L_2 , hence from (48), we have

$$\begin{aligned}
L_2 &= \lambda_2 R_2 \\
&= 5.3 \quad .
\end{aligned}$$

Thus on the average there are 5 or 6 Q_1 requests being served by the CPU and awaiting service.

The number of requests in Q_2 is determined by the feedback from Q_1 of those which are no longer eligible for Q_1 . Let p_{23} be the probability that a Q_1 request is feedback to Q_2 then we have

$$p_{23} = p_{22} + p'_{23}$$

where

$$p_{22} = P(Y_k \geq .05)$$

that is, the probability that a Q_1 request has been serviced for 50 m-seconds without interruption, and

$$p'_{23} = P(X_k \geq .4)$$

that is, the probability that a Q_1 request has received at least .4 seconds of CPU time.

Now

$$\begin{aligned} p_{23} &= \frac{1}{35} + e^{-2} \\ &= .16 \quad (\text{approx.}) \end{aligned}$$

That is, 16% of all Q_1 requests are feedback to Q_2 to receive further processing.

It has been shown by Takacs [52], that for a stationary queueing system with Poisson input density, λ , and exponential service time, μ , with one server, the output is also Poisson with parameter λ . Therefore the output from Q_1 is Poisson with density $\lambda_2 = 4$.

5.4 Analysis for Q_2 , that is, Priorities ≥ 3 .

As shown in the previous section, the total input to this queue, Q_2 , is Poisson with density $p_{23} \lambda_2 = .64$. The application of the model to this queue is, as has been stated, only an approximation where general service time distributions are considered. However, it is applicable when exponential service time is assumed as is done here. It is very unlikely that all users' tasks will be runnable during any average time period. Hence as a more realistic input rate of runnable users to Q_2 let us use estimates of $\frac{1}{16}$ and $\frac{1}{8}$ of the total input $\lambda_3 = p_{23} \lambda_2$ to Q_2 .

Since the formulas for the various moments were given, in general, in Sections 5.1 and 5.2 and specifically for Q_1 in Section 5.3, only the actual values will be stated for the corresponding numerical cases. The two cases considered will have input densities, λ_3 , of .04 and .08 and, for a numerical value of ρ_3 , CPU traffic intensity for priority 3 , let us use .10 .

Case I. $\lambda_3 = .04$, $\mu_3 = .4$

- i) The average service time is 2.5 secs.
- ii) The average number of quanta required per request is 72 .
- iii) There will be an increase of 566% in time to complete service of a Q_2 request.
- iv) The average length of a busy period of priority ≤ 3 is 50 secs.
- v) The average waiting time before being served is 55 .

- vi) The average flow time is 72 secs.
- vii) The average queue length (waiting or being served) varies between 2 and 3 .

Case II. $\lambda_3 = .08$, $\mu_3 = .8$

- i) The average service time is 1.25 .
- ii) The average number of quanta received is 36 .
- iii) The average increase in time needed to completely service a Q_2 request is 566% .
- iv) The average length of a busy period of priority ≤ 2 is 25 secs.
- v) The average waiting time before a Q_2 request receives its first service is 38 .
- vi) The average flow time is 46.3 .
- vii) The average queue length (waiting or being served) varies between 3 and 4 .

CHAPTER VI

CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY

6.1 Conclusions.

As is necessary in any analytical study of a system as complex as a time-sharing system, it has been necessary to combine several interacting factors to be treated as one variable. There are, however, certain vital characteristics of the physical situation which must be brought out in a model if it is to be a fair representation of the former.

A model, whose exactness has been limited by the development of Queueing Theory and, in particular, Priority Queueing Theory, has been developed to show the priorities of the various requests and interrupts in the CP/67 system. The model assumes a general service time distribution for Q_1 so that, in this sense, it is quite general. Thus observations, not available at present but which will be obtained by Gatha [27], could be used to get more accurate results. Meanwhile we have assumed values in Sections 5.3 and 5.4 based on analyses done by various researches such as Oppenheimer and Weizer [38]. These do not necessarily give an accurate representation of the CP/67 system but are still included as a basis of numerical study.

These values tend to indicate a 90% usage of the CPU time which may be considered to be slightly optimistic. However the results for Q_1 seem quite in keeping with the real situation. The quantum size of 50 m-seconds causes a Q_1 request to be feedback 5 times on

the average; a quantum size of, say, 100 m-seconds, as is used at some installations, would cause the request to be feedback an average of 3 times. The increase of 5% in the time needed to service a Q_1 request to completion implies that for Q_1 the servicing of interrupts does not cause substantial delay. On the basis of this model and with the values assumed, the average queue length is 5 or 6 and hence the maximum length of Q_1 may be fixed at 6. (In CP/67 this maximum length is at present 9.) The variances of the parameters, except in the case of the busy periods, have been relatively small, e.g., a value of .05 for the completion time.

The exponential service time distribution has been assumed only in order to obtain numerical values for the second moment; otherwise, the model is valid for interactive or Q_1 requests with any arbitrary service time distribution. We therefore conclude that this model provides a good description of the scheduling algorithm of the CP/67 system, and that, when measurements are available from Gatha's [27] study, reasonable predictions can be made of the different parameters discussed in this thesis.

When this model is applied to Q_2 requests, i.e., requests requiring more than 0.4 seconds, exponential service time distributions have to be assumed. This assumption is necessary because at present the theory of queues has not been sufficiently developed to provide expressions for the output from queues with general service time distributions. In an average time period, there will be only a fraction of Q_2 requests actually seeking service. This fraction cannot be determined at present and hence, to apply the model to Q_2 , we have estimated its value. The

results obtained for the two cases considered show very large values for the waiting time and completion time; the variances too have been quite large. For example, the expected waiting time and its variance are 55 and 3225 respectively.

In view of the apparent success in applying the model to Q_1 and the results obtained when applied to Q_2 , we must conclude that the assumptions for Q_2 were not all valid. Particularly, the input density λ_3 and the average service, a_3 , seem difficult to approximate and hence may be the cause of the failure of the model for Q_2 .

However we conclude that this model is a good representation of the scheduling algorithm of the CP/67 system and that it will be validated when actual measurements are used for the several random variables.

6.2 Suggestions for Further Study.

1. A study such as this depends greatly on the availability of adequate theory to have general applications. Therefore, research can be carried out to determine the output from queues where the service time is not exponentially distributed. Simulation may be used to verify any results obtained in such a study.

2. In the CP/67 system whilst the interrupts (e.g. Paging, I/O etc.) are preemptive, higher priority tasks, e.g. Q_1 requests, cannot preempt lower priority ones, e.g. Q_2 requests, but must await an interrupt. A small study could be undertaken to determine, generally, in what

cases preemptions should be allowed in queueing systems, and, in particular, whether the existing technique in CP/67 is efficient.

3. The model presented here could be compared with Coffman's Foreground-Background model to determine which would be more efficient for the CP/67 dispatcher.

4. Denning [22] studied the behaviour of a "virtual console" in a multiuser computer environment. A similar study on the virtual machine dedicated to the APL users could provide some indication as to whether that virtual machine should have priority over the others in the CP/CMS system.

BIBLIOGRAPHY

1. Abate, J., Dubner, H. and Weinberg, S.B., "Queueing Analysis of the IBM 2314 Disk Storage Facility", J. ACM, 15.4 (Oct. 1968), pp. 577-589.
2. Arden, B.W., Galler, B.A., O'Brien, T.C., and Westervelt, F.H. "Program and Addressing Structure in a Time-Sharing Environment", J. ACM, 13, 1 (Jan. 1966) pp. 1-16.
3. Baylis, M.H.J., Fletcher, D.G., and Howarth, D.J., "Paging studies made on the ICT Atlas Computer", Proc. IFIP Conference 1968 pp. D113-D118.
4. Belady, L.A., "A study of replacement algorithms for a virtual storage computer", IBM Systems J., 5.2 (1966) pp. 78-101.
5. Bell, C.G., "The Fundamentals of Time-Sharing", Time-Sharing American Data Processing Inc., Detroit, 1967.
6. Chang, W., "Preemptive Priority Queues", Op. Res., 13, 1965 pp. 820-827.
7. Chang, W., "A queueing model for a simple case of time-sharing", IBM Systems J., 5.2, (1966) pp. 115-125.
8. Chang, W., "Congestion Analysis of a Computer Core Storage System", Naval Res. Log. Quart. 15.3 (Sept. 1967) pp. 367-379.
9. Chang, W., "Queues with Feedback for Time-Sharing Computer System Analysis", Op. Res., 16, Vol. 1, 1968, pp. 613-627.
10. Churchill, R.V., Operational Mathematics, McGraw-Hill, N.Y., 1958.
11. Cobham, A., "Priority Assignment in waiting line problems", Op. Res. 2, 1954, pp. 70-76.
12. Coffman, E.G., "Analysis of Two Time-Sharing Algorithms Designed for Limited Swapping", J. ACM, 15.3 (July 1968), pp. 341-353.
13. Coffman, E.G., "Analysis of a Drum I/O Queue under Scheduled Operation in a Paged Computer System", J. ACM, 16.1 (Jan. 1969) pp. 73-90.
14. Coffman, E.G., Kleinrock, L., "Feedback Queueing Models for Time-Shared Systems", J. ACM, 15.4 (Oct. 1968), pp. 549-576.
15. Coffman, E.G., Kleinrock, L., "Computer Scheduling Methods and Their Countermeasures", Proc. AFIPS, 1968, SJCC, pp. 11-21.

16. Coffman, E.G., Schwartz, J.I., and Weissman, C., "A general purpose time-sharing system", Proc., ACM, 22nd Nat'l. Conf. (1967).
17. Control Program - 67/Cambridge Monitor System Manual, IBM Cambridge Scientific Center, Mass. 1968.
18. Conway, R.W., Maxwell, W.L., and Miller, L.W., Theory of Scheduling, Addison-Wesley Pub. Co., Mass. 1967.
19. Corbato, F.J. Daggett, M., and Daley, R.C., "An Experimental Time-Sharing System", Proc. AFIPS, 1962, SJCC 21, pp. 335-344.
20. Denning, P.J., "Effects of Scheduling on File Memory Operations" Proc. AFIPS, 1967, SJCC 30, pp. 9-22.
21. Denning, P.J., "The working set model for program behaviour" Comm. ACM, 11.5 (May 1968), pp. 323-333.
22. Denning, P.J., "A Statistical Model for Console Behaviour in Multiuser Computers", Comm. ACM, 11.9 (Sept. 1968), pp. 605-612.
23. Dennis, J.B., and Daley, R.C., "Virtual memory processes and sharing in Multics", Comm. ACM, 11.5 (May 1968), pp. 306-312.
24. Estrin, G., and Kleinrock. L., "Measures, models, and measurements for time-shared computer utilities", Proc. ACM, 22nd Nat'l. Conf., 1967, pp. 85-96.
25. Fine, G.H., Jackson, C.W., and McIsaac, P.V., "Dynamic program behaviour under paging", Proc. ACM, 21st Nat'l. Conf. (1966).
26. Fine, G.H., and McIsaac, P.V., "Simulation of a Time-Sharing System", SDC Report SP - 1909, 1964.
27. Gatha, A.K., "Statistical Evaluation of CP/67 - A Time-Sharing System", Master's Thesis, U. of Alberta.
28. Gaver, D.P., Jr., "A waiting line with interrupted service, including priorities", J. Roy. Stat. Soc. Ser. B, 24, 1962.
29. IBM/360 - 2314 Direct Access Storage Facility. IBM Manual No. A26-3599-2, IBM Corp., 1965.
30. Iverson, K.E., A Programming Language, John Wiley, 1960.
31. Kesten, H. Runnenberg, J.T., "Priority in Waiting Line Problems", Proc. Akad. Wet. Amst. 60, 1957, pp. 312-336.

32. Kleinrock, L., "Time-Shared Systems: A Theoretical Treatment", J. ACM, 14.2 (April 1967), pp. 242-261.
33. Martin, J., Design of Real-Time Computer Systems, Prentice-Hall, Englewood Cliffs, N.J., 1967.
34. McGee, W.C., "On Dynamic Program Relocation", IBM Systems J. 4.3, 1965, pp. 184-199.
35. Miller, R.G., Jr., "Priority Queues", Ann. Math. Stat. 31, 1960, pp. 86-103.
36. Moberg, L.V., "An executive system for on-line programming on a small-scale system", Proc. AFIPS, 1967, FJCC, Vol. 31, pp. 243-254.
37. Mielsen, N.R., The Analysis of General Purpose Computer Time-Sharing Systems, Ph. D. Thesis, Stanford University, 1966.
38. Oppenheimer, G., and Weizer, N., "Resource Management for a Medium-Scale Time-Sharing Operating System", Comm. ACM, 11.5 (May 1968), pp. 313-322.
39. Penny, J.P., "An Analysis, Both Theoretical and By Simulation of a Time-Shared Computer System", Computer J. 9, 1966-67, pp. 53-59.
40. Phipps, T.E., "Machine Repair as a Priority Waiting Line Problem", Op. Res. 4.1, 1956, pp. 76-85.
41. Pyke, T.N., Jr., "Time-shared computer systems", Advances in Computers, Vol. 8 (Alt. F. Ed.) Academic Press, N.Y. 1967.
42. Ramamoorthy, C.V., "The Analytic Design of a Dynamic Look Ahead of Program Segmenting System for Multiprogrammed Computers", Proc. ACM, 21st Nat'l. Conf. (1966), pp. 229-239.
43. Saaty, T.L., Elements of Queueing Theory, Mc-Graw Hill Book Co., N.Y., 1961.
44. Scherr, A.L., An Analysis of Time-Shared Computer Systems, MIT Press Cambridge, 1967.
45. Schrage, L.E., Some Queueing Models for a Time-Shared Facility", Ph. D. Thesis, Cornell University, Feb. 1966.
46. Shemer, J.E., "Some Mathematical Considerations of Time-Sharing Scheduling Algorithms", J. ACM, 14.2 (April 1967), pp. 262-272.
47. Stanger, B., Personal Communication Relating to his Master's Thesis, 1969.

48. Staudhammer, J., Combs, C., and Wilkinson, G., "Analysis of Computer Peripheral Interference", Proc. ACM, 22nd Nat'l. Conf., 1967, pp. 97-102.
49. Stephan, F.S., "Two Queues under Preemptive Priority with Poisson Arrival and Service Rates", Op. Res. 6, 1958, pp. 399-418.
50. Strachey, C., "Time-Sharing in Large Fast Computers", Proc. International Conf. on Information Processing, UNESCO (June 1959), Paris, B.2.19, pp. 336-341.
51. Takacs, L., "A Single Server Queue with Feedback", Bell System Technical Journal, 42.2 (March 1963), pp. 505-519.
52. Takacs, L., Introduction to the Theory of Queues, Oxford University Press, N.Y., 1962.
53. Takacs, L., "Priority Queues", Op. Res. 12, 1964, pp. 63-74.
54. Totschek, R.A., "An Empirical Investigation into the Behaviour of the SDC Time-Sharing System", SDC Report SP-2191, 1965.
55. Wegner, P., "Machine Organization for Multiprogramming", Proc. ACM, 22nd Nat'l. Conf., 1967, pp. 135-150.
56. Welch, P.D., "On Preemptive Resume Priority Queues", Ann. Math. Stat. 35.2, 1964, pp. 600-612.
57. White, H., and Christie, L.S., "Queueing with Preemptive Priorities or with Breakdown", Op. Res. 6, 1958, pp. 79-95.
58. Wood, T., "A Generalized Supervisor for a Time-Shared Operating System", Proc. AFIPS, 1967 FJCC, Vol. 31, pp. 209-214.
59. Ziegler, J.R., Time-sharing data processing systems, Prentice Hall, N.J., 1967.

APPENDIX A

SOME PROBABILITY DISTRIBUTIONS USED IN QUEUEING THEORY

The Poisson Distribution

The most common arrival process used in queueing theory is the Poisson process. In addition to providing easy analysis, the Poisson approximates the number of arrivals in a given time period in many physical situations. Let us consider the following random process:

- i) if a system is in state E_n , ($n = 0, 1, 2, \dots$) at time t , let the probability of at least one arrival in the interval $(t, t+\Delta t)$ be

$$\lambda \Delta t + o(\Delta t) \quad \text{where } \lambda > 0 ;$$

- ii) if the system is in state E_n , ($n = 0, 1, 2, \dots$) at time t , let the probability of the number of arrivals exceeding one in the interval $(t, t+\Delta t)$ be

$$o(\Delta t) .$$

Let $P_n(t)$ be the probability that exactly n units are in the system at time t .

Then we obtain

$$P_n(t+\Delta t) = P_{n-1}(t)[\lambda \Delta t + o(\Delta t)] + P_n(t)[1 - \lambda \Delta t - o(\Delta t)] .$$

Dividing by Δt , we have

$$\frac{P_n(t+\Delta t) - P_n(t)}{\Delta t} = \lambda [P_{n-1}(t) - P_n(t)] + \frac{o(\Delta t)}{\Delta t} [P_{n-1}(t) - P_n(t)]$$

and as $\Delta t \rightarrow 0$

$$\frac{d}{dt} P_n(t) = \lambda \{P_{n-1}(t) - P_n(t)\} ,$$

since

$$\frac{o(\Delta t)}{\Delta t} \rightarrow 0 \quad \text{as} \quad t \rightarrow 0 .$$

When $n = 0$, set $P_{-1}(t) = 0$, then

$$\frac{d}{dt} P_0(t) = -\lambda P_0(t) ,$$

and

$$P_0(t) = e^{-\lambda t} .$$

Hence we get

$$P_n(t) = \frac{(\lambda t)^n e^{-\lambda t}}{n!} , \quad n = 0, 1, 2, 3, \dots ,$$

which is a Poisson process with parameter λt , λ being the rate of arrival.

When an arrival distribution is Poisson with parameter λt , we can show that the interarrival time is exponentially distributed (i.e. the probability distribution of time between consecutive arrivals is exponential with parameter λ). Note that if an arrival has just occurred, the next will occur in time less than t with the following probability

$$P[\text{one or more arrivals up to time } t] = P[\text{time between arrivals} \leq t]$$

$$= \sum_{n=1}^{\infty} \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

$$= 1 - e^{-\lambda t},$$

which is the distribution function of the interarrival time. By differentiation we obtain the density function of the interarrival-time random variable (r.v.) as

$$\frac{d}{dt} (1 - e^{-\lambda t}) = \lambda e^{-\lambda t},$$

which is the density function of the exponential distribution. Some properties of the Poisson process are:

- i) it is 'memoryless', that is, probability estimates of the time to the next arrival are independent of the time at which the last arrival occurred;
- ii) the sum of several independent Poisson r.v.'s has a Poisson distribution with parameter equal to the sum of the parameters of the separate Poisson r.v.'s;
- iii) it is called a random process, since the arrival of a unit in an interval $[0, t]$ occurs completely at random (i.e. it is uniformly distributed in the interval).

The Erlang Distribution.

The Erlang Distribution, which is sometimes used to represent service time, has less variability about the mean than the exponential distribution and hence it has a smaller standard deviation. Suppose t is the time required for a unit to pass through k servers each with exponential service time and parameter λ . The Erlang density function is the probability density function of t which is given by

$$f(t) = \frac{1}{(k-1)!} (k\lambda)^k t^{k-1} e^{-k\lambda t} \quad t > 0 ,$$

where k and λ are positive parameters of the distribution.

The Erlang distribution function can be easily obtained from this expression and for different values of the parameter k , there is a family of Erlang distributions. The distribution coincides with the exponential distribution of service time when $k = 1$ and for k at ∞ the service time is constant. Empirical service-time distributions can be approximated by an Erlang distribution by using the appropriate value of k .

APPENDIX B

The Laplace Transform of a Distribution Function

Let $H(x)$ be the distribution function of the random variable X i.e. $H(x) = P\{X \leq x\}$, then its Laplace transform (L.T.) $\psi(s)$ is given by

$$\begin{aligned}\psi(s) &= E\{e^{-sX}\} \\ &= \int_0^{\infty} e^{-sx} dH(x)\end{aligned}\tag{i}$$

where $s \geq 0$ or $\text{Re}(s) \geq 0$ if s is complex. The L.T. of $H(x)$ exists if the integral in (i) converges for some value of s ; otherwise it does not exist. Through the use of the Stieltjes integral, the L.T. can be obtained for distribution function which are continuous, discrete, or a combination of these two types.

The following properties of L.T.'s associated with distribution functions are used in this thesis:

- i) There is a one-to-one correspondence between a distribution function and its associated L.T. so that one uniquely determines the other. The inverse L.T., $L^{-1}\{\psi(s)\}$, may be recovered through tables or inversion formulae.
- ii) The k^{th} moment of X , $E(X^k)$, is given by

$$E(X^k) = (-1)^k \left[\frac{d^k}{ds^k} \right] \psi(s) \quad s=0$$

if and only if a finite limit $\psi^{(k)}(0)$ exists. This is the moment generating property of the distribution.

iii) If $L^{-1}\{\psi(s)\} = H(x)$ and $L^{-1}\{\theta(s)\} = F(x)$, where $F(x)$ is also a distribution function, then

$$\begin{aligned} L^{-1}\{\theta(s)\psi(s)\} &= \int_0^x H(x-u)dF(u) \\ &= F * H \end{aligned}$$

where $F * H$ is called the convolution of $F(x)$ and $H(x)$.

Applying the property recursively, the L.T. of the sum of n identically and independently distributed random variables each with L.T. $\theta(s)$ is $[\theta(s)]^n$. This is referred to as the L.T. of the n -fold convolution of $F(x)$ with itself.

A more complete analysis of the Laplace Transform can be found in several mathematical texts e.g. Churchill [10].

APPENDIX C

The Distribution of Some Random Variables Encountered in Both the Preemptive-Resume and Head-of-Line Disciplines.

In both the head-of-the-line and the preemptive-resume service disciplines, the actual service time is the time the task or request is interacting with the service facility. With the preemptive-repeat service discipline, on the other hand, there is wastage of processing whenever the service of the request is interrupted [18]. Hence certain distributions are identical or approximately equal for the preemptive-resume and head-of-the-line disciplines. The distributions, which are discussed, are those of the busy periods, completion time, waiting time and queue size.

To show that the distribution of the busy period is independent of the service discipline, let us consider a lemma due to Welch [56].

Lemma. Consider a $G/G/1$ queueing process (i.e. arbitrary inter-arrival and service time distributions in a single server system), subject to the following conditions:

- i) the server is busy whenever there are requests in the queue;
and
- ii) the total time any customer is in contact with the server is his service time.

For this queueing process, the lengths of the busy period of the server are independent of the service discipline. This result is true even if the service is divided into quanta.

Proof. Let τ_n be the arrival time of the n^{th} arriving request, χ_n be his service time (for $n = 0, 1, 2, \dots$) and $n(t)$ be the occupation time of the server at time t is the sum of the remaining service time for the current request and the service times of all requests in the queue at time t .

Let us consider the sequence of values $\{n(t)\}$ for $t \geq 0$, then $n(t)$ increases in magnitude by χ_n at points τ_n , and, in between these points it either decreases with slope of -1 if $n(t) > 0$, or remains unchanged if $n(t) = 0$, regardless of the service discipline. The lengths of the busy periods are determined by $n(t)$ and are therefore independent of the service discipline.

The busy period of priority k clearly depends only on those requests with priorities $\leq k$ and it terminates when there are no requests with priorities $\leq k$. Thus as Miller [35] observed, the distributions of the busy period for the same are identical for the preemptive-resume and head-of-the-time disciplines.

Gaver [28], who introduced the concept of completion time, showed that its distribution is the same for the preemptive-resume and head-of-the-line disciplines. The random variable, C_n , the completion time of the n^{th} arriving request, he expressed as

$$C_n = S_n + \sum_{i=0}^N D(i),$$

where S_n is the request's service time, $D(i)$ the direction of the i^{th} interruptions. Now whether these higher priority requests interrupt the current service of a k^{th} priority request or are handled after its

completion, does not affect the completion time of the k^{th} priority requests. Hence the distributions of completion time are identical.

As far as the distributions of waiting time are concerned Miller [35] showed that they are the same for both preemptive-resume and head-of-the-line disciplines.

Gaver [28], in his analysis, obtained expressions for the queue sizes for the preemptive-resume and head-of-the-line. He observed that they differ only in one term. For the preemptive-resume discipline,

$$E(N) = \rho + n_0$$

and for the postponable (head-of-the-line) case

$$E(N) = \rho \frac{E(S)}{E(C)} + n_0$$

where ρ is the traffic intensity, $E(S)$ the expected service time, $E(C)$ ($= E(S)[1 + \nu E(D)]$) the expected completion time.

Thus in the head-of-the-line discipline for a stationary process, the expected queue size is less than for the preemptive-resume discipline by the term

$$\begin{aligned} \rho \left[1 - \frac{E(S)}{E(C)} \right] &= \rho \left[1 - \frac{1}{1 + \nu E(D)} \right] \\ &= \rho \frac{E(D)}{1 + \nu E(D)} \end{aligned}$$

where $E(D)$ is the expected duration of the interrupts (higher priority requests) and ν is the rate of interrupt arrivals i.e. Λ_{k-1} . Since this term is relatively small we can consider the expected queue sizes of the two service disciplines to be approximately the same.

B29917